RUNDECK

Clearing the Way for SRE in the Enterprise

The full chapter 10 from O'Reilly Media's "Seeking SRE" book.

DAMON EDWARDS

Table of Contents

01

Introduction

“Sounds great, but how would that ever work here?”

Do you work in a medium to large enterprise? Did this question cross your mind while reading this book? Know that you are not alone.

Changing how an organization does it’s operations is difficult at any scale. However, it is in the enterprise where the challenges and roadblocks to change will often seem like insurmountable mountains.

Change your tools? Complicated, but doable at any size.

Teach individuals new skills? Difficult, slow work but there are known paths for everyone to follow.

Fundamentally change how your organization works? This is where you hit that metaphorical mountain in all but the smallest of organizations.

Fear not, moving from a classic enterprise operations model to an SRE model is doable. Companies are doing it right now as you read this book.

This chapter is for enterprise leaders who are seeking to transform their traditional operations organizations to SRE. You will learn how to identify and clear the obstacles that, if left unaddressed, would otherwise undermine your SRE transformation.

You will learn how to identify and clear the obstacles that, if left unaddressed, would otherwise undermine your SRE transformation.

This chapter comes from a compilation of knowledge gained while working with large enterprises to transform their operations organizations. These folks have taken the uncharted path to SRE so that your journey may be easier.

First, I will examine the systemic forces that will be standing in your way. Second, I will highlight techniques for clearing those obstacles so that you can start and sustain your SRE transformation.

How should you measure success? Improved reliability, improved MTTD/MTTR[1], improved organizational agility, and fulfilled effective colleagues. What is failure? Ending up with some new “SRE” job titles, but not much else changing.

02

Toil, the Enemy of SRE

Toil is the hidden villain in the journey to SRE. In organizations who have already internalized the SRE way of working, spotting and eliminating toil is a natural reflex. In organizations who are coming from a traditional operations culture, spotting and eliminating toil is a organization-wide skill that often needs to be learned.

What is toil? Vivek Rau from Google articulates[2] this answer well, “Toil is the kind of work tied to running a production service that tends to be manual, repetitive, automatable, tactical, devoid of enduring value, and that scales linearly as a service grows.” The more of these attributes a task has, the more certain you can be that the task should be classified as toil.

Being classified as toil doesn’t mean that a task is frivolous or unnecessary. On the contrary, most organizations would grind to a halt if the toil didn’t get done. The value of toil is often a point of confusion amongst traditional enterprise operations teams. To some, manual intervention in the running of services is their job description.

Just because a task is necessary to deliver value to a customer, that doesn’t necessarily mean that it is value-adding work. For people who are familiar with Lean manufacturing principles, this isn’t dissimilar to Type 1 Muda[3] (necessary, non-value adding tasks). Toil may be necessary at times, but it doesn’t add enduring value (i.e. a change in the perception of value by customers or the business).

Instead of your SREs spending their time on non-value-adding toil, you want them spending as much of their time as possible on value-adding engineering work. Also pulling from Vivek Rau's helpful definitions, Engineering work can be defined as the creative and innovative work that requires human judgement, has enduring value, and can be leveraged by others. [Fig 11-1]

Toil	Engineering Work
Lacks Enduring Value	Builds Enduring Value
Rote, Repetitive	Creative, Iterative
Tactical	Strategic
Increases With Scale	Enables Scaling
Can Be Automated	Requires Human Creativity

Fig. 11-1 - Comparing characteristics of toil and engineering work

Working in an organization with a high ratio of engineering work to toil feels like, metaphorically speaking, everyone is swimming towards a goal. Working in an organization with a low ratio of engineering work to toil feels more like you are treading water, at best, or sinking, at worst.

A goal of "no toil" sounds nice in theory but, in reality, that isn't attainable in an ongoing business. Technology organizations are always in flux, and new developments (expected or unexpected) will almost always cause toil.

The best we can hope for is to be effective at reducing toil and keeping toil at a manageable level across the organization. Toil will come from things you already know about but just don't have the time or budget to automate initially (e.g., semi-manual deployments, schema updates/rollbacks, changing storage quotas, network changes, user adds, adding capacity, DNS changes, service failover, etc.). Toil will also come from any number of the unforeseen conditions that can cause incidents requiring manual intervention (e.g., restarts, diagnostics, performance checks, changing config settings, etc.).

Toil may seem innocuous in small amounts. Concern over individual incidents of toil is often dismissed with a response like “nothing wrong with a little busy work.” However, when left unchecked, toil can quickly accumulate to levels that are toxic to both the individual and the organization.

For the individual, high-levels of toil lead to:

- Discontent and a lack of feeling of accomplishment
- Burnout
- More errors, leading to time-consuming rework to fix
- No time to learn new skills
- Career stagnation (hurt by a lack of opportunity to deliver value-adding projects)

For the organization, high-levels of toil lead to:

- Constant shortages of team capacity
- Excessive operational support costs
- Inability to make progress on strategic initiatives (the “everybody is busy, but nothing is getting done” syndrome)
- Inability to retain top talent (and acquire top talent once word gets out about how the organization functions)

One of the most dangerous aspects of toil is that it requires engineering work to eliminate it. Think about the last deluge of manual, repetitive tasks you experienced. Doing those tasks doesn’t prevent the next batch from appearing.

Reducing toil requires engineering time to either build supporting automation to automate away the need for manual intervention or enhance the system to alleviate the need for the intervention in the first place.

Engineering work needed to reduce toil will typically be a choice of creating external automation (i.e., scripts and automation tools outside of the service), creating internal automation (i.e., automation delivered as part of the service), or enhancing the service to not require maintenance intervention.

Toil eats up the time needed to do the engineering work that will prevent future toil. If you aren't careful, the level of toil in an organization can increase past a point where the organization

won't have the capacity needed to stop it. If aligned with the Technical Debt metaphor, this would be "engineering bankruptcy." **[Fig. 11-2]**

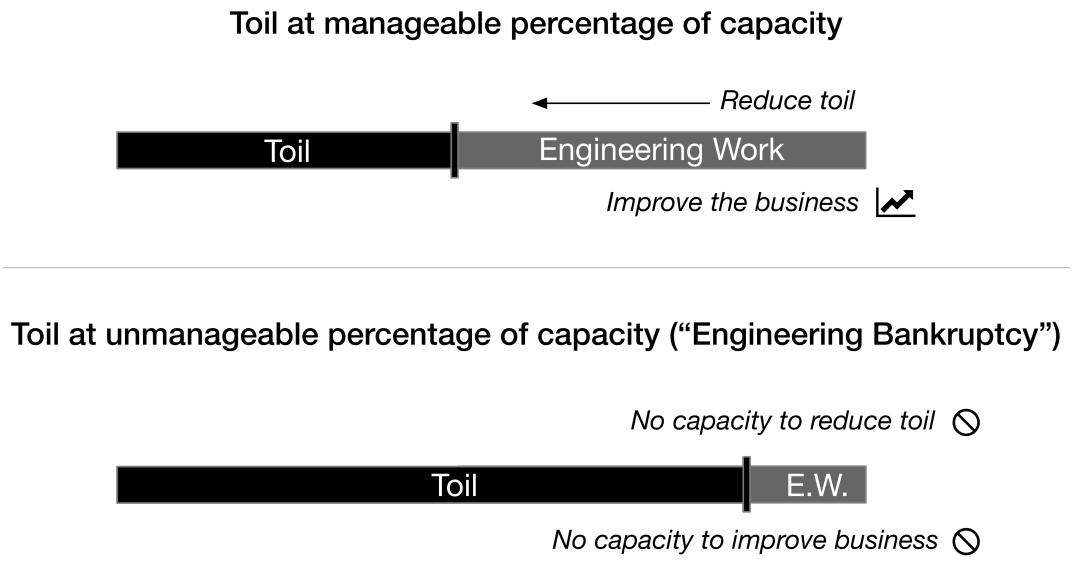


Fig. 11-2 - Excessive toil consumes a team's capacity to perform engineering work to both improve the business and reduce toil.

The SRE model of working — and all of the benefits that come with it — depend on teams having ample capacity for engineering work. If toil eats up that capacity, the SRE model can't be launched or sustained. An SRE perpetually buried under toil isn't an SRE, they are just a traditional long-suffering sysadmin with a new title.

03

Toil In the Enterprise

Enterprises are fertile ground for toil. First, when it comes to the concept of toil, legacy operations management philosophies were either blind ("Everybody looks busy. Great efficiency!") or indifferent ("Why are you complaining about these headaches? I pay you to have headaches."). Second, the high level of organizational complexity found in an enterprise creates toil and gets in the way of efforts to reduce toil.

For this discussion, let's define an "enterprise" as any company that has had the historical success needed to accumulate a significant amount of legacy (culture, organization, process, and technology).

Enterprises have a distinct "look." From a business perspective, you'll find multiple business lines, each born or acquired during different eras that had unique context and underlying assumptions. From a technology perspective, you'll find multiple generations of platforms and tools — some brand new, some old and evolving, some orphaned — that all need to hang together to provide services to customers.

It is good to keep in mind that nothing in an enterprise lives in isolation. What you are working on is dependent on others. What others are working on is dependent on you. In classic architectures, these dependencies are fixed and obvious. In modern architectures, these dependencies are often dynamic and abstracted away, but still exist. At the human level, incentives, budgets, policies, beliefs, and cultural norms are all intertwined across the various ends of an enterprise.

This interconnectedness makes eliminating toil significantly more challenging in the enterprise. The toil of your team's own making can be eliminated with straightforward engineering work. But what about all of the toil that is due to conditions or systems that exist in other parts of the organization? Eliminating it is out of a team's control unless that team can affect solutions across organizational boundaries — which anyone with enterprise experience knows is no trivial feat.

Toil that is partially, or wholly, out of a team's control is especially dangerous. It pushes the team closer to that bankruptcy threshold where toil crowds out all engineering work. This is a common cause of the anti-pattern where sysadmin teams are rebranded "SRE teams" but the lack of engineering blocks transformation beyond the new name.

04

Silos, Queues, and Tickets

Once you established that excessive toil will prevent an enterprise from shifting to an SRE model, it logically follows that you are going to have to work across organizational boundaries in order to effectively contain toil. However, working across organizational boundaries is one of the great challenges in enterprise IT.

Working across organizational boundaries is difficult because of a confluence of silo effects, request queues, and the most venerable of all IT operations workhorses, tickets.

Silos Get In the Way

The silo metaphor is first attributed to Phil S. Ensor, who used it in 1988[4] to describe the organizational challenges at his employer, Goodyear Tire. Since then, the concept of ‘silos’ has been discussed by the Lean manufacturing movement and the DevOps movement. Some people mistakenly think "silos = teams,” but in reality, silos don't have that much to do with organizational structure. The idea of a silo really has everything to do with how a group(s) works within an organization.

In simple terms, a group is said to be ‘working in a silo’ when its members are working in a disconnected manner from other groups (whether they know it or not). [Fig 11-3] When spotting silos, look for situations where a group is working in a different context from

other groups, their work is coming from a different source than other groups (i.e., different backlogs), and the group is working under different incentives or priorities (and often part of a different management chain). It is almost certain that this group is working in a silos and experiencing any number of symptoms: bottlenecks, slow handoffs, miscommunication, tooling mismatches, delivery errors, excessive rework, and conflict (usually the finger-pointing type).

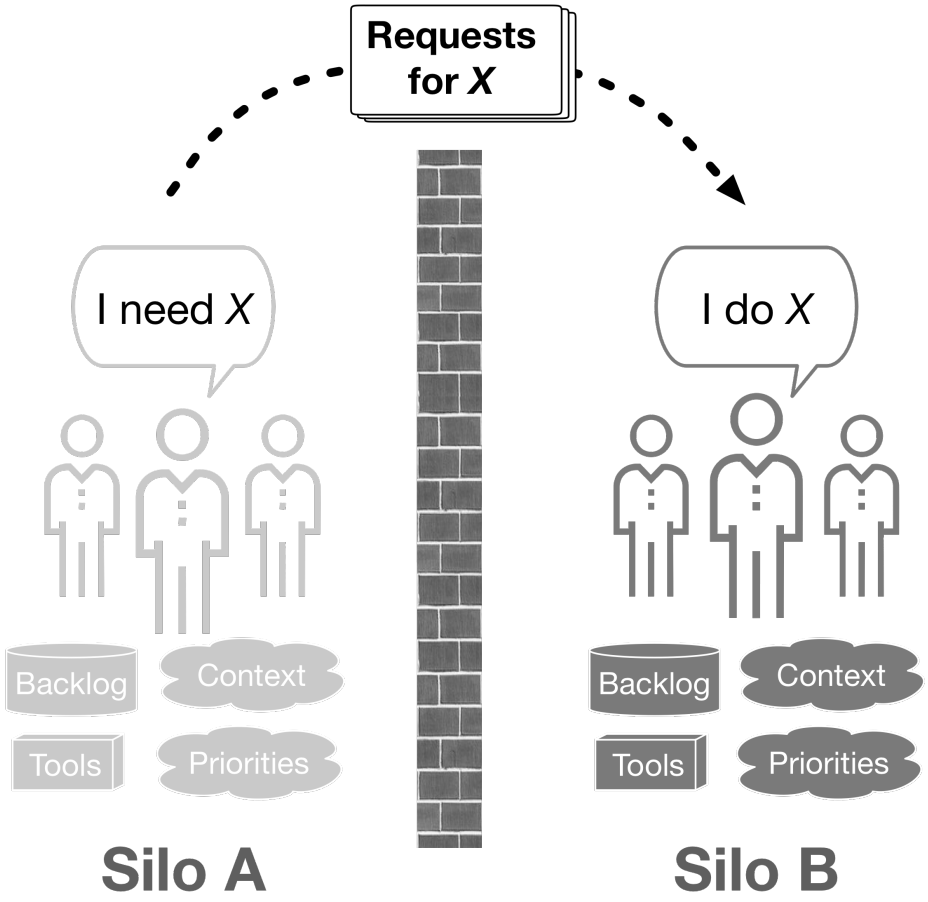


Fig. 11-3 - Silos describe a way of disconnected working rather than a specific organizational structure.

If you've worked in operations in an enterprise, you probably recognize this condition. Some might even ruefully describe it as "the way things have always work". It is endemic to the classic operations model where you have lots of specialist teams divided by functional expertise and use ticket systems and a heavy reliance on project management to coordinate work and push it through the organization.

Operations teams don't set out to work in silos or suffer the consequences of silos. It is usually the natural byproduct of traditional management philosophies based on the human urge to "optimize" large-scale efforts by sorting people according to functional specialization, grouping like-with-like, and then incentivizing them to look inward and optimize.

Problematic handoffs (too slow, incorrect, lots of rework, etc.) are the most commonly cited problem that can be attributed to silo effects. This makes sense as silos only become a problem when you need something from someone outside of the silo (or someone outside of the silo needs something from you).

And remember, in the enterprise, nothing lives in isolation. Doing anything significant usually means information and work is going to have to cross one or more organizational boundaries.

What goes wrong when work has to pass between siloed groups? It usually has to do with mismatches: [Fig 11-4]

- **Information mismatches** – The parties on either side of the handoff are working with different information or are processing the information from different points of view, leading to an increase in errors and rework (i.e. repeat work due to previous errors).

- **Process mismatches** – The parties on either side of the handoff are following either different processes or processes that are nominally the same but take a different approach and produce results not expected by the other party. Timing and cadence mismatches between parts of a process that take place in different silos also lead to an increase in errors and rework.

- **Tooling mismatches** – An increase in errors and rework is seen when different parties on either side of silo boundaries are using different tooling or tooling that isn't setup to connect seamlessly. When the work needs to be translated on the fly by a person moving information and artifacts by hand from one tool to another, delay, variance, and errors are bound to be introduced into the process.

- **Capacity mismatches** – Bottlenecks and delays are a result when the amount or rate of requests coming from one side of the silo boundary exceeds the capacity of those fielding the requests. Requests levels often exceed or are below expectations which has the ripple effect of disrupting the planning and flow of work for other parts of the organization.

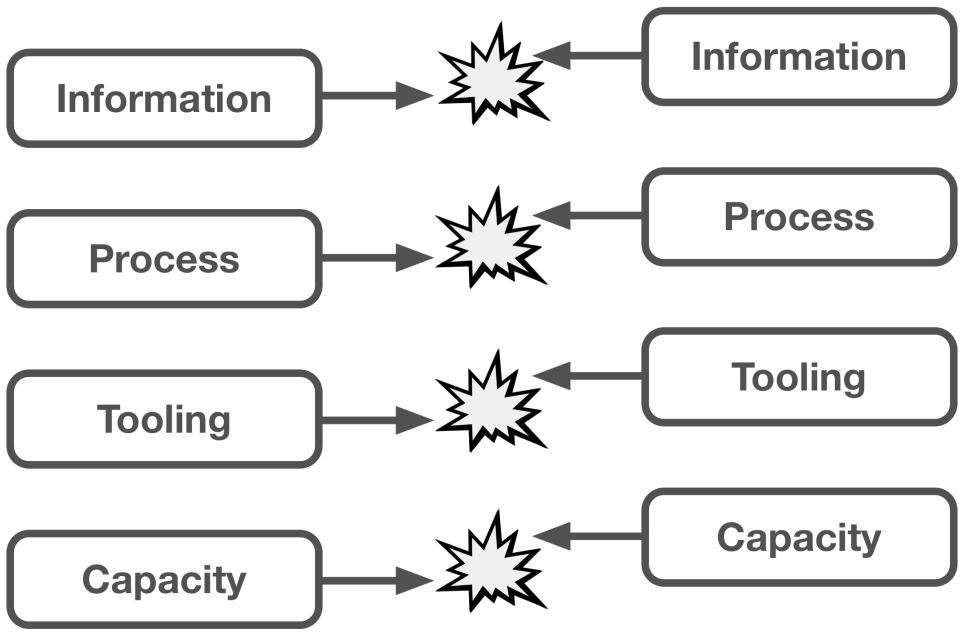


Fig. 11-4 - Handoffs between silos are problematic due to mismatches

Ticket-Driven Request Queues Are Expensive

For decades, the goto countermeasure for dealing with the handoff problems caused by silos is to insert a request queue to govern the handoff (usually a ticket system).

On the surface, request queues might seem like an orderly and efficient way to manage work that crosses the divides of an organization. However, if you look under the surface you will find that request queues are a major source of economic waste for any business. Let us look at the following list, created by noted author and product development expert, Donald G. Reinertsen, that catalogs the negative impact of queues.[5]

Queues create: [Fig 11-5]

- **Longer Cycle Time** – Queues increase cycle time as it takes longer to reach the front of a large queue than than a small one. Even small delays can exponentially compound within a complex interdependent system like an enterprise IT organization.

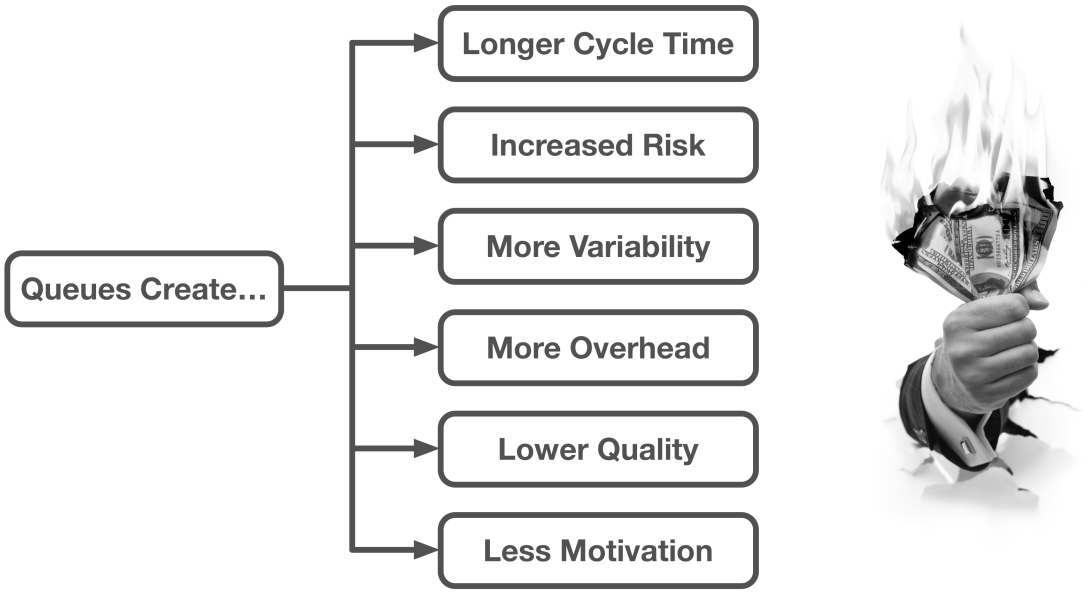
- **Increased Risk** – Queues increase the time between request and fulfillment which in turn increases likelihood of the context of the original request changing (a race condition). If a problem does arise, the requester is now in a different mental position (often working on something else) from where they were when they made the request.

- **More Variability** – Longer queues lead to high levels of utilization and higher levels of utilization amplify variability. This leads to longer wait times and a higher likelihood of errors.

- **More Overhead** – Queues add a management overhead for managing the queue, reporting on status, and handling exceptions. The longer the queue, the more these overhead costs grow in a compounded manner.

- **Lower Quality** – Queues lower quality by delaying feedback to those who are upstream in the process. Delays in feedback causes the cost of fixing problems to be much higher (e.g. bugs are easier to fix when caught sooner) and often means that additional problems of a similar origin have been created before the first negative feedback arrives.

Less Motivation – Queues have a negative psychological effect by undermining motivation and initiative. This is due to queues (especially longer queues) removing the sense of urgency and immediacy of outcomes from the requestor's work. If you don't feel the impact and don't see the outcome, it's human nature to grow negatively disconnected from the work.



Adapted from Donald G. Reinertsen, The Principles of Product Development Flow: Second Generation Lean Product Development

Fig. 11-5 - Queues have been proved to be economically expensive

Looking at the science behind the impact of queues, it is difficult to justify inserting queues all over your organization. It doesn't make sense to willing inject longer cycle times, slower feedback, more risk, more variability, more overhead, lower quality, and decreased motivation into your organization.

Despite this, what is the most common method of managing work inside IT operations organizations? Request queues in the form of ticket systems.

When an organization manages the interaction between people working in other silos via tickets, you are undermining people's capacity for value-adding engineering work both directly (more wait time, more overhead, more disconnects, more errors) and indirectly (more toil). Worse still, much of this lack of capacity and increased toil is outside of their capability to fix because the cause is in another silo.

In the coming sections, I'll discuss how to eliminate silos, queues, and tickets — and how to avoid their harmful effects where they can't be eliminated.

05

Take Action Now

Hopefully, the first part of this chapter made the case that SRE demands a change to the fundamental conditions prevalent in legacy enterprise operations organizations. The following sections lay out steps you can take to clear the obstacles to an SRE transformation.

Like any transformation, you should favor a continuous improvement approach. Your organization is a complex system. Big bang transformations of complex systems are risky and have a poor record of success. No matter how much you plan upfront for an SRE transformation, it will likely go differently once you start moving. Encourage your team to embark on a series of steady, deliberate actions.

The suggestions in the rest of this chapter should not be looked at as prerequisites to taking action or a definitive formula to follow (i.e. “the one true way”). These are patterns and lessons to apply on a continuous basis. Gather your new SRE team. Empower them to start transforming how they work. Empower them to reach across organizational boundaries and collaborate on improvements that will benefit everyone. Favor action and continuous improvement.

06

Start by Leaning on Lean

If you are going transform how your operations organization works, you might as well leverage a proven body of transformational knowledge. The Lean manufacturing movement has produced a wealth of design patterns and techniques[6] that can be applied to improve any work process. In particular, it is the principle of Kaizen (roughly translates to “continuous improvement”), born from the Toyota Production System, that speeds transformations and drives an organization's ability to learn continuously. To bring Kaizen to an organization, there is a method called Kata, also based on Toyota Production System.

Kata is an excellent methodology to apply to the challenge of eliminating toil, silos, and request queues. Kata encourages organizations to look at the end-to-end flow of work and methodically experiment until the desired outcome is reached. Teams are encouraged to see beyond their work silo and think holistically about problems. Kata will help you identify what stands in the way of your goals and then stay aligned as you iterate towards those goals.

It is important to point out that Kata is about teams improving themselves. Lasting performance improvement comes from when people know how to fix problems as part of their day-to-day work. One-off projects or external help might provide a specific benefit at a specific point in time. However, enterprises don’t stand still. New challenges, large and small, will arise all the time.

SRE teams are meant to do engineering work to improve the reliability and operability of systems and reduce toil (freeing up more time for engineering work). SRE teams, by definition, have to be learning teams who can spot and fix problems as part of their day-to-day work. Kata is an excellent reference to teach new SRE teams how to think and work like that.

The Kata approach is already becoming a common practice in Agile and DevOps efforts looking to improve development and delivery processes. However, operations processes are rarely deemed worthy of the effort, especially in legacy operations cultures. This oversight is unfortunate as Operational quality is equal in value to any other quality measure in your organization. Even ad-hoc operations processes (e.g., one-offs due to an incident or similar type of event) are processes that are worthy of study.

If you are considering the move to SRE, then it should be assumed that your organization already knows the value of investing in improving operations work. If not, a conversation on value needs to happen among both technical and business leaders in your company.

There are many books, presentations, and other resources available to take you deep into the practice of Kata. The works of Mike Rother[7] and John Shook[8] are highly recommended. However, part of the beauty of Kata is that you don't need much knowledge to get started and begin seeing benefits.

Here is an overview of the Kata process in an operations context:

1. ***Pick a direction or challenge.*** This is your higher-level directional goal. Where do you want to go as an organization? What is the business value you are there to maximize? What is the ideal state of operations work at your company?

   It is critical that there is consensus on what the value of operations is to the business and what performance and reliability levels need to be to maximize that value. The entire organization (from frontline engineers to leadership) should know why they are embarking on this SRE transformation and how will it improve the business.

   The answers don't have to be overly detailed or address how you are going to get there. However, if the answers end up sounding like general platitudes or a vague mission statement, that is a failure (e.g., "Delight customers with highly available and speedy services"). Keep trying until people have a decent notion of where the organization needs to go and why. There should be some recognition of both measurable operational outcomes (e.g., reducing incidents, improving response times, increased frequency of change, etc.) and desired business outcomes (e.g., increased sales, higher net promoter score, etc.)

2. ***Grasp the current condition.*** Build a clear understanding of how things work today. Take an end-to-end view of the process you want to improve and strive to understand how it is actually done today. Why does it happen this way? What is needed to get it done? Who is needed to get it done? What gets in the way? When does it go wrong?

Be sure to look at each of the types of work in your organization. Project-oriented work is an obvious choice (e.g. systems engineering, environment build outs, etc.). It is equally important to look at incidents (i.e., failure scenarios). Examining incidents as processes might sound odd because incidents rarely follow much of a standard process beyond some communication and information gathering formalities. However, if you look at enough incidents, you will find common patterns. Incidents provide a very enlightening look into how an organization is really 'programmed' to work.

Remember, silos are your enemy. Doing this kind of analysis in private or limiting it to a team of 'experts' delivers minimal value to your organization. Enterprises are notorious for their compartmentalization of information. Very few people will know how how end-to-end processes actually happen — and even they will disagree. You are bound to open a lot of eyes and hear comments like "hmm that's not how I thought that worked" or "I never knew that" from even the longest-tenured employees. That alone is worth the effort.

Do your analysis in the open and encourage participation from as many people as possible. Look upstream and downstream for participants with knowledge to contribute. Invite developers, program managers, and other operations teams. Your transformation to SRE has the best chance of success if as many people as possible have a similar understanding of what needs to change.

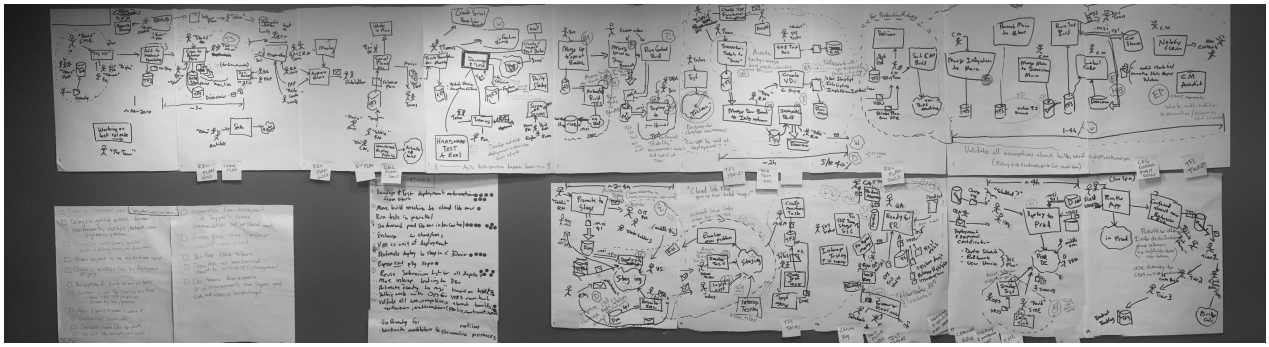


Fig. 11-6. - Artifacts from visual analysis created during a process retrospective session

Visual analysis is a very effective way to get a group of people aligned — assuming you do the visual analysis as a group. In Lean terminology, this is known as 'Going to the Gemba', which roughly translates to 'going to the place where the work happens'. In the physical manufacturing world, this means going to the factory and directly observing the work firsthand.

In IT Operations you can't actually see most of the work happen. Other than some artifacts, the work is all abstractions and individual's mental models. So Going to the Gemba in our context only works if you are getting people together and aligning those mental models the best you can. This alignment is extremely important because those individual mental models are the lens through which every person in your organization evaluates and executes their daily work. [Fig 11-6]

So how do you do this? This visual analysis works best when you look at one process at a time. Pick a process (a specific delivery of project work or a specific incident).

Get people together who have first-hand knowledge of the process (enough to look at it end-to-end). Draw out what actually happened (focus on the flow of work, with people and tools as 2nd-level supporting details). Finally, get the participants to identify and discuss what got in the way. Repeat this with enough instances of the process (or group of related processes) until there is reasonable consensus on how the process happens, why it happens (if it is an incident), and what gets in the way.

There are other resources you can draw on to develop your visualization and analysis technique. Value Stream Mapping[9] and Lean Waste classification[10] are highly useful.

3. ***Establish your next Target Condition.*** This step is where you decide on the next improvement target that your organization will attempt to reach. Again, this is best done as a group. Based on the directional goal determined in step 1 and the current condition uncovered in step 2, what is the next major intermediate step we are going to try to accomplish? This is what the organization is going to focus on, so everyone should be able to articulate what it should be like when the Target Condition is achieved.

   As Mike Rother advises, "Setting the target condition is not about choosing between existing options or best practices. It's about aspiring to new performance. By setting a target condition and trying to achieve it, you learn why you cannot. That's what you work on."

4. ***Experiment towards the Target Condition.*** Get the organization into an iterative rhythm of forming hypotheses (e.g, “If we could stop/start x, it would reduce/increase y by z amount”) testing alternatives (e.g., process, tooling, or org changes), and evaluating the result. If the result moves you towards the Target Condition, implement whatever was tested and continue with other experiments. If you see the Scientific Method in this, you are correct. Repeat the experimentation until you’ve reached the next target condition.

   During step 4 of this process, be sure you revisit step 2 often to make sure everyone still grasps the current condition. Also, make sure you are reviewing the specification of the next target condition often to make sure everyone stays aligned.

   Once you hit the target condition, repeat step 1 (“is the goal still valid?”) and then steps 2-4.

06

Get Rid Of As Many Handoffs As Possible

As was described earlier in this chapter, silos and their accompanying problematic handoffs and expensive request queues do considerable damage to an organization. So perhaps it is obvious that the first strategy to relieve this problem is to get rid of as many silos and handoffs as you can!

Forward-thinking organizations are transforming from a traditional "vertical" structure aligned by functional skill to a "horizontal" structure aligned by value stream or product. The vertical structure is the classic divide-by-function strategy (e.g., Dev, QA, Ops, Net, Sec, etc.). The horizontal structure is comprised of cross-functional teams who can own the entire end-to-end lifecycle of a service.

The idea behind cross-functional teams is that they will handle as much of the lifecycle as possible without needing to hand work off to other teams. There are no significant handoffs or breaks in context, and everyone's work flows from a single backlog aligned by common priorities. Bottlenecks are largely avoided, feedback loops are shorter, and cycle times are quicker. If a problem does arise, it is easier for a cross-functional team to respond and rectify the problem. [Fig 11-7]

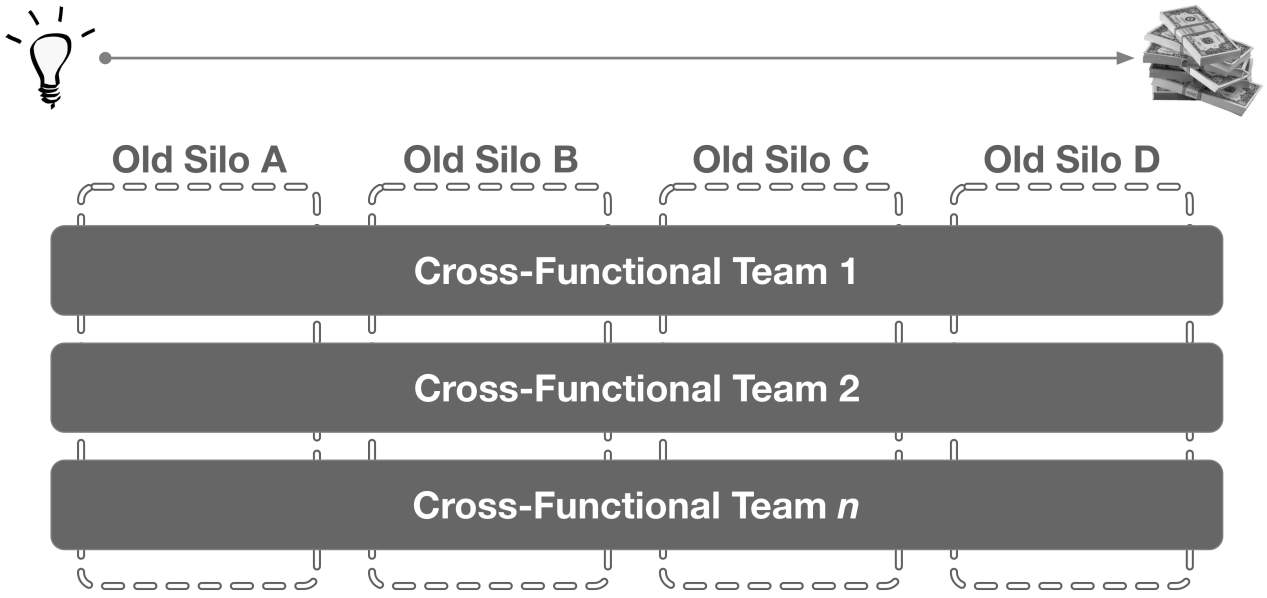


Fig. 11-7 - Cross-functional teams alleviate the need for many handoffs

These cross-functional teams are also often referred to as 'service-oriented teams,' 'product-oriented teams,' or 'market-oriented teams.' These labels emphasize that the goal of these teams is alignment towards a business-recognizable delivery of value to a customer (i.e., a specific customer-identifiable service). This alignment allows for the team members to understand how their work impacts the business and then optimize the team for maximizing customer value (rather than functional efficiency).

While the idea of cross-functional teams appears simple, implementation is a significant structural and cultural change for enterprises.

To understand why this is such a significant shift for enterprises, look no further than the rules of corporate accounting. Admittedly, this is an area that most engineers thought they would never have to delve. However, some basic concepts provide a window into some of the most fundamental forces that shape an enterprise.

Project-based funding is often the primary flow of money in enterprise IT. Once a business need is identified there is a process to define a project and fund it with a specific budget. After funding, the project will proceed through the various functional silos of the IT organization until the project reaches production and is deemed 'complete.' The next iteration is generally looked upon as a new project with a distinct budget.

Project-based funding increases the pressure on operations in several ways:

First, the proliferation of services and bespoke infrastructure generates a significant amount of toil for operations. When an organization is project-oriented, teams move from project to project, often leaving a trail of new software and new infrastructure behind. The movement of teams to new projects deprives the teams of learning from operational feedback. Also, as discussed earlier in this chapter, the constant creation of net new software and infrastructure (even if built following documented patterns) creates new technical debt and errors from unknown conditions. These behaviors are continuous sources of toil for the teams on which operational responsibility is dumped.

Second, in the drive to hit a project's finish line, operational concerns are not always adequately addressed before being pushed to production. 'Delivering' the project at or under the budget is

what the delivery teams are judged on. It is only human nature that operational concerns like stability, scalability, deployability, configurability, observability, and manageability tend to get superficial treatment or are often the first thing 'cut' in a time crunch. In the worst cases, these operational concerns are not considered at all. While the 'high-value' development resources are quickly recycled and re-attached to projects, an operations team has to catch each project and own reliability and scale. In a traditional model, this is a continuous stream of toil that usually requires adding more headcount. However, even assuming the operations teams are equipped with the skills to do engineering work in an SRE model, these teams are still in perpetual catch-up mode, and engineering bankruptcy is always a risk if toil levels get too high.

Third, with projects being the primary funding vehicle, operations budgets are viewed as mostly operations and maintenance expenses, or 'OpEx' in accounting lingo. OpEx is the budget category that gets the most scrutiny and cost controls since it directly impacts the current year's profitability. Teams whose funding largely comes from OpEx are naturally susceptible to management impulses for organization-by-function and efficiency mandates, which both encourages siloed working.

Toil mostly appears around operations and maintenance work, which is OpEx. Generally, all of the project based funding is CapEx. Engineering work to build or improve a system is usually CapEx as it is improving an asset (and can be amortized over several years, so the negative impact on current year profit is smaller). A team funded out of an OpEx budget and being managed for efficiency, often won't have the budget (or the general charter) to engage in significant engineering work.

To move to an SRE model, you do not have to be an accounting expert. However, it does pay dividends to be acutely aware of how the money flows in your company. If you want your move to SRE to be more than a change in job titles, you are going to have to make the case that the SRE team should be funded to do engineering work and attached to teams who own the service lifecycle, from inception to decommissioning. The move from project-based funding to product-based funding (with cross-functional teams) is already gaining traction in Agile and DevOps discussions. If this idea is making headway in your organization, try to leverage it for you SRE transformation.

Two common patterns are shaping up in enterprises experimenting with cross-functional teams and SRE.

The first is to have SREs (and other functional roles) join dedicated product teams. This creates cross-functional teams that can own a service from inception to decommissioning. Development and ongoing operations all happen from within these cross-functional teams. From an enterprise perspective, this is often seen as a radical departure from traditional organization models. Sometimes you may hear this referred to as the “Netflix” model.

The second is to have SRE remain a distinct organization. Development teams have SRE skills and initially will own the full lifecycle of a service. Once the service has reached a certain level of performance and stability, there is a formal handoff to an SRE team who has embedded development skills. This SRE team manages the availability and scalability of the service according to performance agreements made with the development teams.

If further changes made by the development team drop the performance of the service below the agreed upon levels, there is a mechanism to return more operational responsibility to the development team. From an enterprise perspective, the general shape of this model seems the most familiar. However, how work happens in this model is still a radical departure from traditional operations models. Sometimes this is referred to as the “Google” model.

Of course, employees from both Netflix and Google will be quick to point out that there is a lot more nuance to either model. However, these descriptions provide a possible high-level starting point for thinking about how to devise a model that works best for your company’s unique conditions.

The options available to you will likely be constrained by how the rest of your company wants to operate. For example, changing organizational structure to move to product aligned teams entirely will require buy-in from at least development and product management (which will likely need broader business discussions). Staying with a traditional development and operations organizational divide has the benefit of not upending the entrenched political structures within your company, but at the same time, those political structures can reinforce the old ways of working and unintentionally undermine improvement efforts.

The visualization techniques described in step 2 of the Kata process (‘Grasp the current condition’) can help with discussions on org structure changes. Not only does visualization help people understand the flow of work, but it also helps them see how org structure impacts that flow of work.

In any model you choose, there is still bound to be culture shock when the walls between traditionally development-only teams and operations-only teams are broken down. SREs play an essential role in bringing operations skill and experience to previously development-only teams. In the physical fitness word there is an expression, “Great abs are made in the kitchen, not in the gym”. Likewise, “Great operations starts in development, not in production.” SRE’s play the critical role in bringing operations skills and discipline to previously development-only teams.

07

Replace Remaining Handoffs With Self-Service

There are always cases in the enterprise where you cannot put all of the required skills and knowledge into cross-functional teams. For practical, financial, or political reasons, most companies operating at significant scale cannot avoid having functional specialist teams. Networks, platforms, security, and data management are common areas where enterprises rely on centralized teams because of either not enough specialists to go around or the need (perceived or real) to centralize control.

The presence of these specialist teams means that handoffs that can't be avoided during the normal flow of work through the organization. Either you need something from one of them to get your work done, or you are on one of these specialist teams where everyone needs something from you.

Dependencies between services (a fact of life in the enterprise) require teams to consume services from other teams and make operational requests of those teams (e.g., configuration changes, health checks, performance tuning, deployment coordination, adding accounts, etc.). The creates yet another set of unavoidable handoffs.

As was discussed earlier, whenever there is a handoff point, and work has to move from one team's context to another, there is an

opportunity for silo effects to take hold and create problems.

Since you cannot get rid of all handoffs, you have to apply techniques and tooling to mitigate the negative impact of those handoffs. Deploying the traditional solution of ticket-driven request queues is expensive, toxic to the organization, and should only be a last resort.

Instead, the go-to solution should be to deploy self-service capabilities at those handoff points. These self-service capabilities should provide pull-based interfaces to whatever was previously needed to be done by someone on the other end of a request queue (e.g., investigating performance issues, changing network/firewall settings, adding capacity, updating database schemas, restarts, etc.).

The point of self-service is to stay out of the way of people who need an operations task completed. Rather than having someone fill out a ticket and sitting in a request queue, you give them a GUI, API, or command line tool to do it themselves, when they need to do it. This capability eliminates wait time, shortens feedback loops, avoids miscommunication, and improves the labor capacity of the teams that previously had to field those requests (freeing them from repetitive requests so they can focus on value-adding engineering work).

08

Self-Service Is More Than a Button

The idea of self-service isn't new. However, the traditional approach to self-service is to have a privileged operations team create somewhat static 'buttons' for less privileged teams to push (e.g., push button deployment for new .war files). There are a limited number of scenarios where this static approach works. Also, it still leaves the higher privileged team as the bottleneck because they are the ones who need to build — and maintain — the button and the underlying automation. This maintenance burden alone limits the amount of static self-service capabilities an operations organization can expose.

In order to maximize the effectiveness of self-service, the ability to both the define and execute automated procedures needs to be provided. Of course constraints need to be put into place to enforce security boundaries and help prevent mistakes, but providing the ability to define and execute delivers the most value. [Fig 11-8]

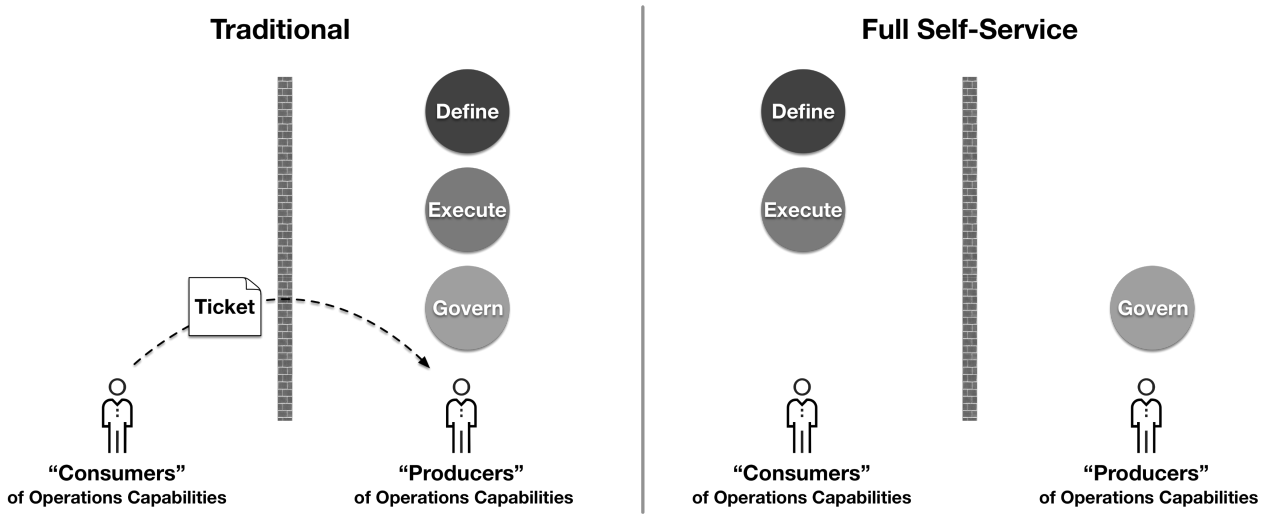


Fig. 11-8 - Traditional ticket-driven request fulfillment vs full self-service

For example, consider the EC2[11] service from Amazon Web Services (AWS). The ability to press a button and get a running virtual machine was interesting. However, the ability to control your own destiny by making your own machine images (AMIs) and configuration was revolutionary. It empowered individuals and allowed teams to decouple and move at their own pace. However, it isn't unfettered access. Users are constrained for security reasons by both AWS and their own self-selected security policies[12]. Users are also constrained in their choices to provide 'guardrails' to keep users of the system from making some types of mistakes or impacting performance of other users. In this example, the ability to define and execute automation is pushed to end users and governance is shared by the operators (AWS in this case) and end users.

To maximize the value of self-service in the enterprise, replicate this pattern of moving the ability to define and execute automation to wherever in the organization it will improve the flow of work.

09

Self-Service Helps SREs in Multiple Ways

Within an enterprise, effective self-service capabilities are a boost to SRE efforts. The following is a list of some of the benefits.

Reduces toil - With few exceptions, fielding repetitive requests is toil. Having effective self-service capabilities can help SREs reduce toil by quickly turning around automation to reduce those repetitive requests. Often repetitive requests do not follow the same pattern each time, and this can undermine an SRE's ability to set up reusable automation. If you can set up the right primitives and then give requesters the ability and permission to create their own automated procedures, you can create self-service for an even broader set of repetitive requests. Examining which self-service processes end up being built can also indicate to both SREs and developers where their current engineering efforts should be focused to reduce the need for future intervention.

Alleviates security and compliance concerns - Security and regulatory compliance are unavoidable facts of life in the enterprise. Whether it is organizational scar tissue from past problems, a response to industry fears, or a directive from an auditor, your SRE transformation will have to work within existing security and compliance requirements. Trying to introduce a new operations model and question existing security or compliance policies is not advisable. Pick your battles.

Self-service capabilities can provide both a system of record for operations activity and a point of policy enforcement. The same self-service mechanisms can be used within teams and across team boundaries. This will give you both the ability to track operational activity to meet compliance requirements and a way to safely expand the distribution of access privilege. By doing so, SREs can work across a wider scope of your infrastructure than was previously permitted. This also enables your traditionally non-operations colleagues to participate in operations activity without opening tickets for others to do it for them.

Separation of duties is a standard requirement of most enterprises. No matter if it is to comply with specific regulations (e.g., PCI DSS Requirement 6.4[13]) or to satisfy more general regulatory controls, separation of duties can interfere with a team's plans to take cross-functional ownership of the development, testing, and operations of a service.

Self-service tooling can help by providing a mechanism through which a person in a development role can create a procedure that can be vetted and quickly executed by someone in an operations role. Also, those in privileged operations roles can create pre-approved, limited self-service capabilities that can be executed on-demand by those in non-privileged roles (e.g. development or QA roles). Auditors can often be persuaded that this form of self-service still meets a separation of duties requirement since the operations role sets up the procedure, grants specific access, and receives the logs and notification of its use.

Security concerns are also the culprit behind many of today's repetitive requests that can be labeled as toil. Purely for security reasons, people in are currently being forced into queues and are

waiting for someone to do something for them. Self-service, done correctly, gives the requesters the ability to take action themselves. Security postures can be maintained through fine-grained access control, full logging, and automated notifications.

Improves incident response - Self-service capabilities are helpful for capturing team's best practices as checklists and automated runbooks. When responding to incidents, checklists can improve both individual and team performance. Setting up checklists as automated runbooks not only encourages consistency but also allows the running of the checklists — and the watching of the output — to be a group activity.

Maximizes the value of standard services and infrastructure - It is considered a best practice for SRE teams to use some of their engineering time build and maintain standard infrastructure (e.g., platforms, environments, etc.) and operations services (e.g., deployment systems, observability, etc.). The better the self-service capabilities are, the more an organization will be able to leverage these standard components and services.

10

Operations as a Service

If you examine companies who both employ an SRE style of operations and are regarded as high-performers by their peers, you will find that they have often built custom purpose tooling to enable self-service capabilities. For example, look at Netflix's combination of Spinnaker[14], Winston[15], Bolt[16]. These were originally purpose-built tools developed from the ground up for Netflix's focused, purpose-built organization. Enterprises will probably find that they need more generic self-service capabilities to embrace the heterogeneity that comes from decades of acquisition and accumulation.

Operations as a Service is a generic and deceptively-simple design pattern for creating generic self-service operations capabilities. The basic idea of Operations as a Service[17] is that it is a platform for safely distributing the ability to both define and execute automated procedures. [Fig 11-9]

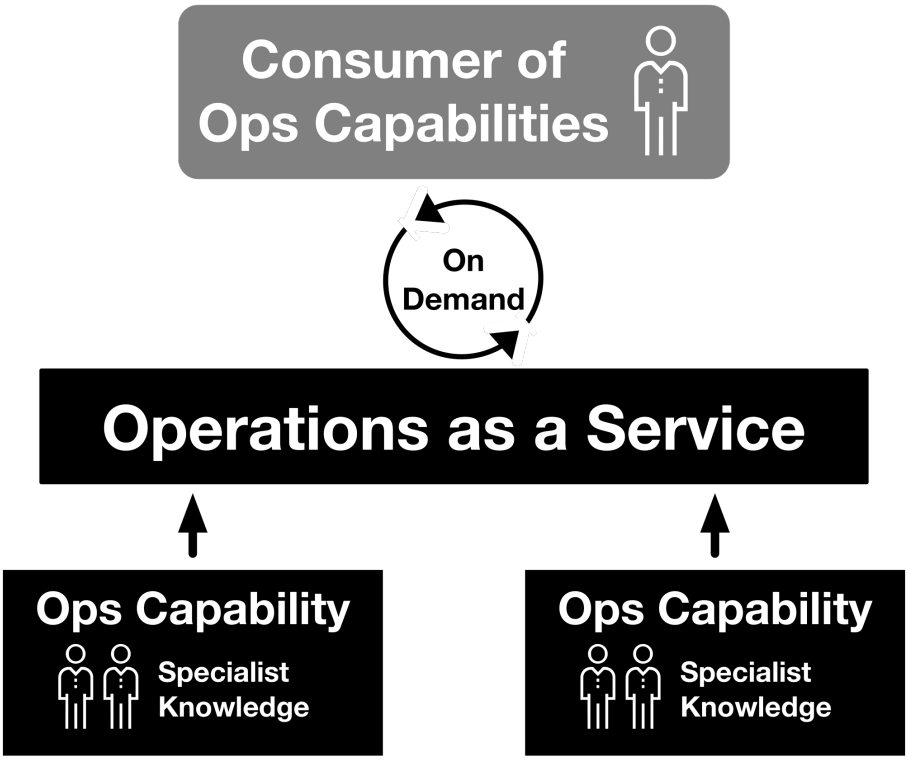


Fig. 11-9 - Overview of Operations as a Service Design Pattern

Critical to the success of this design pattern is the requirement that the platform is both lightweight and work with any popular scripting language or tool. Forcing teams to standardize on one language or automation framework just isn't realistic given the heterogeneous nature of modern enterprises. In fact, it is not just unrealistic; it can actually slow an organization down. Teams need to be able to use the automation languages and tools that they want (or inherited) while allowing for other tools to orchestrate procedures across those underlying frameworks and languages.

Access control and audibility are also critical to the success of this design pattern. For any solution to thrive in the enterprise, ultimate control needs to remain with people and teams who are deemed to have a higher level of access privilege.

Operations as a Service efforts have the best chance of exceeding expectations when paired with monitoring and observability projects. Implementation projects tend to focus heavily on automation. However, once the project progresses, many organizations discover that visibility into operational health, state, and configuration is lacking. Metaphorically speaking, they are distributing the keys to the car without giving people the capability to see where there are going. To avoid this problem, put equal emphasis from the beginning on both “the view” and “the do.”

The Operations as a Service design pattern should be compatible with any operations operating model. Whether you are moving to cross-functional teams or staying with closer to a traditional development and operations organization divide, developing your organization’s self-service capabilities will pay dividends. [Figs 11-10, 11-11]

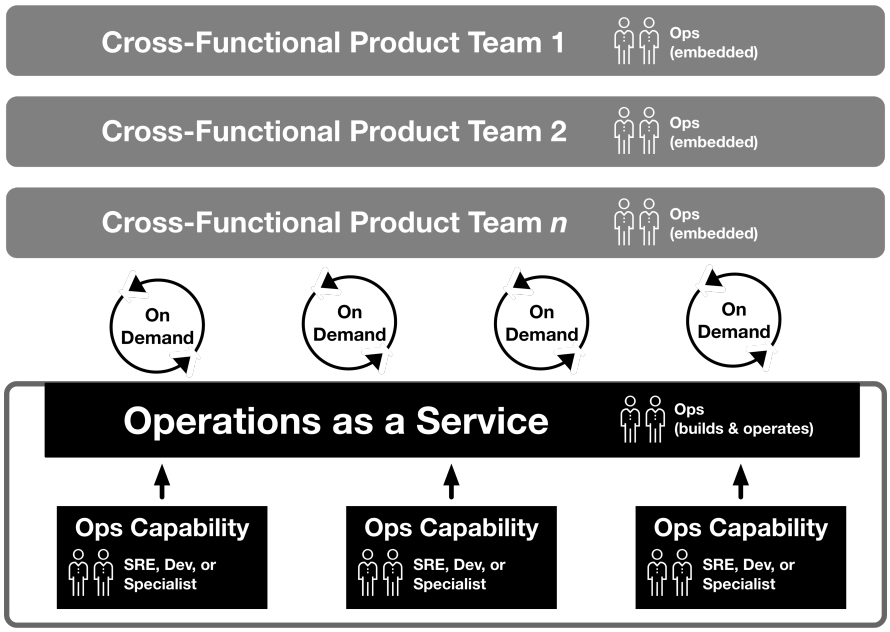


Fig 11-10 - Operations as a Service design pattern with a cross-functional teams organizational model.

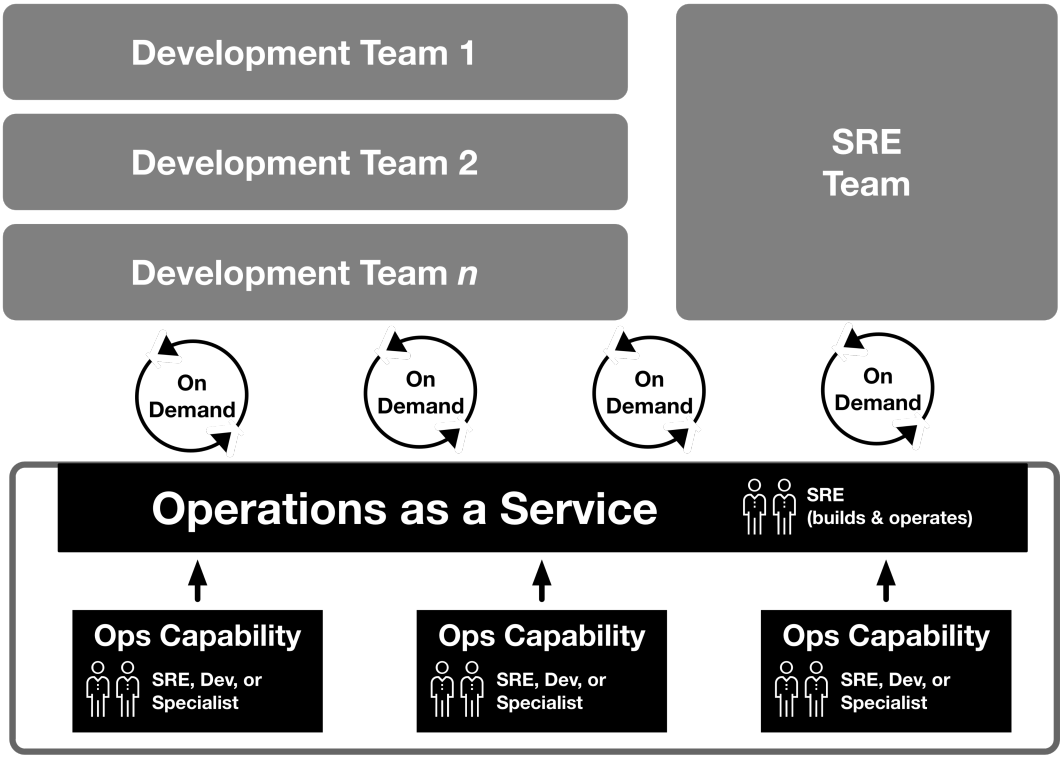


Fig. 11-11 - Operations as a Service design pattern with a more traditional Development and Ops/SRE split organizational model.

11

Error budgets, Toil Limits, and Other Tools for Empowering Humans

One of the more powerful developments to come out of the SRE movement is the popularization of a set of management concepts that formalize the expectations around an organization's operational activity. For those who work in companies that had an SRE model from inception, the following ideas might be self-evident. For those who work in traditional enterprise IT operations, these ideas often highlight how much of a departure the SRE model is from traditional operations beliefs and practices.

Error Budgets - There are often tensions in an organization over how much risk is tolerable for a particular service and who is responsible when the service level objectives are not met. In traditional enterprise divides, the product end of a company is incentivized to go faster, and the operations end is incentivized to avoid downtime and other performance problems. This is the type of mismatch in incentives that encourages silos to form. How do you keep both interests aligned? How do you keep all roles investing in both speed and reliability?

Error Budgets is a framework for measuring — and utilizing — allowable risk. Specifically, it is the gap between theoretically perfect reliability and an acceptable service level objective agreed

upon by business and technology stakeholder. Error Budgets provide a framework to negotiate with the business on how much failure is acceptable and still be able to meet the needs of the business.

It is not an accident that the metaphor of a budget is used. Budgets are a representation of currency that is available to be spent. That is the same with Error Budgets. Developers and SREs can use that budget in attempts to move the business forward. If a service has a small Error Budget, developers and SREs must be more conservative to favor stability. If a service has a bigger Error Budget, Developers and SREs can be more aggressive and favor speed and production experimentation. Like other types of budgets, the negotiation is about how to best spend it (and in some cases, don't spend it at all). [Fig 11-12]

What happens if the Error Budget is exceeded? Error Budgets are attached to a service, not a particular role. All roles involved with the service must respect the budget. For example, if the Error Budget is violated due to aggressive or problematic change, those in development roles need to adapt their behavior (including often taking over more operational responsibility) and adapt the service to perform within the error budget allotted.

This is a sharp contrast to the traditional business-mandated Service Level Agreements (SLA) found in enterprise IT operations. If those traditional SLAs were broken, operations (in a service provider role) incurred the penalty and development — already on to their next project — usually did not. Also, the concept of a service level indicator (quantifiable performance measure), a service level objective (performance target), and the error budget (current amount above the service level objective) is more nuanced and

pragmatic than traditional SLA approaches. Be sure that teams coming from traditional operations cultures understand the differences.

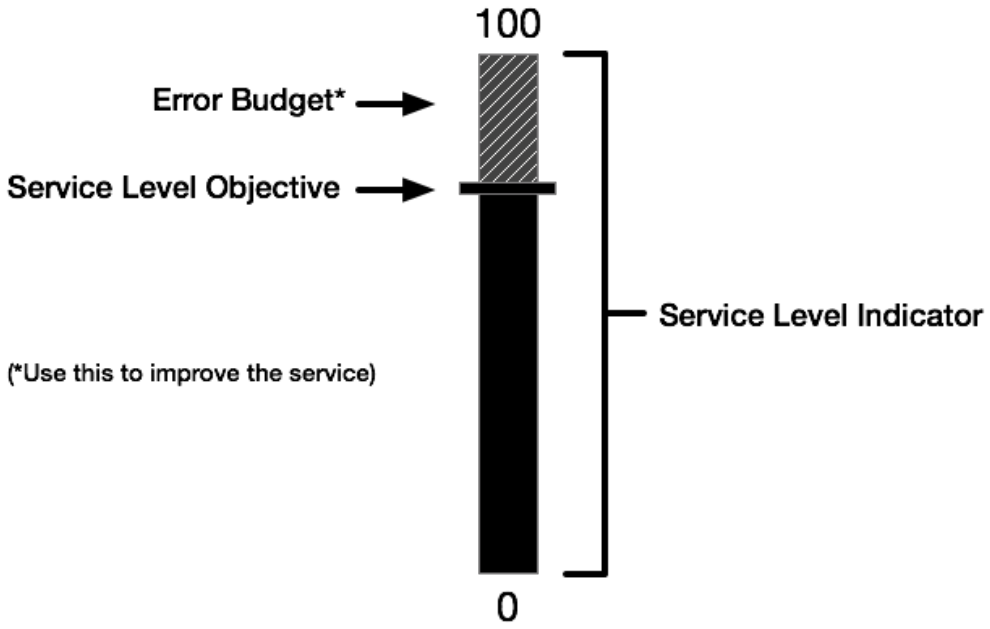


Fig. 11-12 - Error Budget is the difference between perfection and the agreed upon Service Level Objective

Toil Limits - Toil Limits is another concept that challenges traditional operations thinking. I have already covered why toil undermines the SRE function. Defining a limit on the amount of toil that is to be undertaken by an individual SRE or team both makes a statement about priorities and protects an individual or team's capacity to do engineering work.

Toil Limits are also indicators of a team's health. If a team's toil exceeds a predetermined threshold (e.g., Google's default limit of 50% of an engineer's capacity), the organization can swarm to the

find out ways to help rectify the situation. Like Error Budgets, Toil Limits help SREs reach an agreement on expected behavior, provide clear signals when help is needed, and avoid being overrun by non-value adding repetitive work. In traditional IT Operations culture, teams rarely have these types of protections.

The concept of toil is largely absent from traditional enterprise operations culture (despite high levels of what you can now identify as toil). For people working in modern SRE-inspired organizations, toil feels bad. The urge is to find ways to eliminate it, and your colleagues support the effort. In traditional enterprise culture, toil is, at best, considered a 'nice to fix' item and, at worst, just accepted.

When introducing toil limits, education will be needed to both socialize the concept of toil and to get your people to understand why toil is destructive to both the individual and the organization.

Leverage existing enthusiasm for DevOps - While DevOps was once the exclusive domain of web-scale startups, DevOps has become an accepted ideal in most enterprises. Born in 2009, DevOps is a broad cultural and professional movement focused on "world-class quality, reliability, stability, and security at ever lower cost and effort; and accelerated flow and reliability throughout the technology value stream."[18]

There is quite a bit of overlap between the goals of DevOps and SRE. There is also quite a bit of overlap between the theoretical underpinnings of DevOps and SRE. Benjamin Traynor Sloss, the Google leader who first coined the term SRE and presided over the codification of Google's SRE practices, sees a clear overlap between DevOps and SRE.

As Traynor Sloss puts it: "One could view DevOps as a generalization of several core SRE principles to a wider range of organizations, management structures, and personnel. One could equivalently view SRE as a specific implementation of DevOps with some idiosyncratic extensions."[19]

Within the enterprise, DevOps has been applied most often to the limited scope that starts with software development and moves through the service delivery pipeline (from source code check-in to automated deployment). In these enterprises, the penetration of the DevOps transformation is minimal beyond deployment and the bulk of operations practices have remained unchanged. SRE is an opportunity to leverage the momentum started by DevOps and continue the transformation efforts throughout to the rest of the post-deployment lifecycle.

I recommend looking for DevOps momentum in your organization and aligning your SRE transformation efforts. There are lessons that both can learn from each other. Working both Dev-towards-Ops (DevOps) and Ops-towards-Dev (SRE) will give your company's transformation the best chance of success.

Unify backlogs and protect capacity - The concept of working from a single, well-managed backlog and protecting a team’s capacity did not start with the SRE movement. These are both fundamental Lean concepts for improving the flow of work and are popular in both the Agile and DevOps communities. These concepts are a lot less prevalent in traditional operations cultures but can be extremely useful for managing the work of — and protecting — teams transitioning to SRE.

SRE work by definition is a mix of planned and unplanned work.

This mix of work types is among the most difficult to manage. Planned and unplanned work are different modes of working and do not mix well. Having different backlogs for each type of work makes things even worse. It is like serving multiple masters at the same time. It is easy to be pulled in too many directions or to be completely run over by the competing demands.

Engineers working in traditional IT operations organizations often have multiple backlogs with different sources of work managed in different ways. There is one system through which formal project work comes to them. Then there can be another way the team maintains a backlog and manages engineering work that doesn't bubble up to a formal project. Then, of course, there is a different way that interrupt-driven requests, like incidents, are handled.

Moving each team to a single, unified (planned + unplanned) backlog plays dividends. Rather than the constant tension between valuable planned work and necessary unplanned work, unified backlogs make prioritization and trade-offs clear. Unified backlogs also make it easier to reserve capacity for unplanned work (really another type of "budget).

Kanban is a management methodology that features ideas like unified backlogs and protected capacity. Kanban has been shown to significantly improve the flow of work in organizations with mixed types of work. The writing and presentations of longtime Kanban expert and author, Dominica DeGrandis[20], are a good place to start as she has been one of the pioneers in the application of Kanban to operations organizations.

Psychological Safety and Human Factors - It may now seem like common sense that optimizing the performance of your most

important assets, your people, is critical to the success of any technology business. However, this was not always the case. If you have been in the IT operations industry long enough, you have seen traditional operations cultures that treated its people like interchangeable 'cogs.' If a cog wore out, it must not have been tough enough! If the machine broke, there must be a cog to blame! Cogs are just parts, so keep looking elsewhere for the cheapest supplier of cogs!

If you want to build a highly effective, fast-moving organization, you need a culture that empowers your people to engage in reasonable risk-taking, bring up bad news to superiors, engage in creativity, and support their co-workers. Psychological Safety and Human Factors are related fields of study that are much broader than IT but have a lot to offer to our industry[21]. From examinations of airplane disasters to medical tragedies, there is a reusable body of knowledge on how to maximize human performance in stressful, complex situations.

12

Join the Movement

We are working during a unique moment in IT operations history. While we often get new technologies and tools, we rarely do we get an opportunity to reshape the structure, behaviors, and culture of operations. This is an opportunity to improve the work lives of operations professionals around the globe and improve outcomes for their employers.

SRE is a continuously-evolving practitioner-led movement. The best part of a practitioner-led movement is that you can be a part of it. Whether it is online or in-person at conferences or meetups, join in.

Like with most of IT, the early adopters and promoters tend to not be from enterprises. Don't let this dissuade you. First, the lessons learned and the principles debated are more generally applicable than you would think. Second, enterprises might not be quick to adopt new practices, but as community acceptance grows, enterprise adoption will certainly follow. Getting involved early with SRE is a way to both prepare your skills and make your company more competitive.

It doesn't matter if you are learning from the experiences of other practitioners or validating your learnings through sharing your experiences, your participation will return more value than the effort you put in.

Now let's roll up our sleeves and get to work.

[1]Mean Time To Detect / Mean Time to Repair: Mean Time To Detect (MTTD) is the average length of time between the onset of a problem and the problem being detected. Mean Time To Repair (MTTR) is the average length of time between the onset of a problem and the resolution of the problem. While these are commonly used operations metrics, their use is not without controversy. The first point of controversy is how precise of a judgement you can make using these metrics to evaluate operational performance. If no two incidents are alike, how valid is looking at an average? The other controversy is over which metric should take priority. Rapid detection can be an indicator of a system better instrumented and better understood, leading to more consistent resolution and better prevention. Rapid repair might indicate better automation or failover capabilities, and repairing/restoring is the ultimate goal in any incident.

[2]https://landing.google.com/sre/book/chapters/eliminating-toil.html

[3]Womack, J. P., & Jones, D. T. (2010). Lean Thinking: Banish Waste and Create Wealth in Your Corporation. Riverside: Free Press.

[4]Ensor, Philip. S. (1988, Spring). The Functional Silo Syndrome. Target. Association for Manufacturing Excellence

[5]Reinertsen, Donald G.. The Principles of Product Development Flow: Second Generation Lean Product Development.

[6] https://www.lean.org/ and https://www.goldratt.com/

[7]http://www-personal.umich.edu/~mrother/The_Improvement_Kata.html

[8]Rother, Mike, and John Shook. Learning to See Value-Stream Mapping to Create Value and Eliminate Muda. Lean Enterprise Institute, 2009. and Shook, John. Managing to Learn: Using the A3 Management Process to Solve Problems, Gain Agreement, Mentor, and Lead. Lean Enterprise Institute, 2010.

[9]Rother, M., & Shook, J. (2003). Learning to see: value-stream mapping to create value and eliminate muda; a lean tool kit method and workbook. Cambridge, MA: The Lean Enterprise Institute.FMJ3V4VakA and http://stella.report

[10] https://itrevolution.com/the-7-wastes-of-devops/

[11]https://aws.amazon.com/ec2/

[12]https://docs.aws.amazon.com/IAM/latest/UserGuide/introduction.html

[13] https://www.pcisecuritystandards.org/documents/PCI_DSS_v3-2.pdf

[14]https://www.spinnaker.io/

[15]https://medium.com/netflix-techblog/introducing-winston-event-driven-diagnostic-and-remediation-platform-46ce39aa81cc

[16]https://medium.com/netflix-techblog/introducing-bolt-on-instance-diagnostic-and-remediation-platform-176651b55505

[17]https://www.rundeck.com/oaas-guide

[18]Kim, G., Debois, P., Willis, J., & Humble, J.(2017). The DevOps handbook: How to create world-class agility, reliability, and security in technology organizations. Portland, OR: IT Revolution Press, LLC.

[19]https://landing.google.com/sre/book/chapters/introduction.html

[20]http://ddegrandis.com/

[21]https://www.youtube.com/watch?v=C

About the Author

Damon Edwards is Co-Founder and Chief Product Officer of Rundeck, Inc. Damon has spent over 15 years working with both the technology and business ends of IT operations and is noted for being a leader in porting cutting-edge DevOps and SRE techniques to large enterprise organizations.

Damon is also a frequent conference speaker and writer who focuses on DevOps and IT operations improvement topics. Damon is active in the international DevOps community, including being a co-host of the DevOps Cafe podcast, an early core organizer of the DevOps Days conference series55

READY FOR SHORTER INCIDENTS AND FEWER ESCALATIONS?

Let's talk.
hello@rundeck.com