SRE **implements** DevOps

DevOps:

A set of principles & culture guidelines that helps to breakdown the silos between development and operations / networking / security

- Reduce silos
- Plan for failure
- Small batch changes
- Tooling & automation
- Measure everything

Site Reliability Engineering:

A set of practices with an emphasis o strong engineering capabilities that *implement* the DevOps practices, an sets a job role + team

- Reduce silos *by shared ownership*
- Plan for failure *using error budget*
- Small batch changes *with focus o stability*
- Tooling & automation *of manual*

Getting started with Site Reliability Engineering (SRE):

A guide to improving systems reliability at production

Hi, I'm Abeer

Abeer Rahman
Engineering Manager, Deloitte
Focusing on Software delivery, Agile & DevOps for clients

Let's shift operations from **reactive** to **proactive**

Topics

Introduction to SRE

Enabling SREs

Budgeting for Failures

Measuring Systems

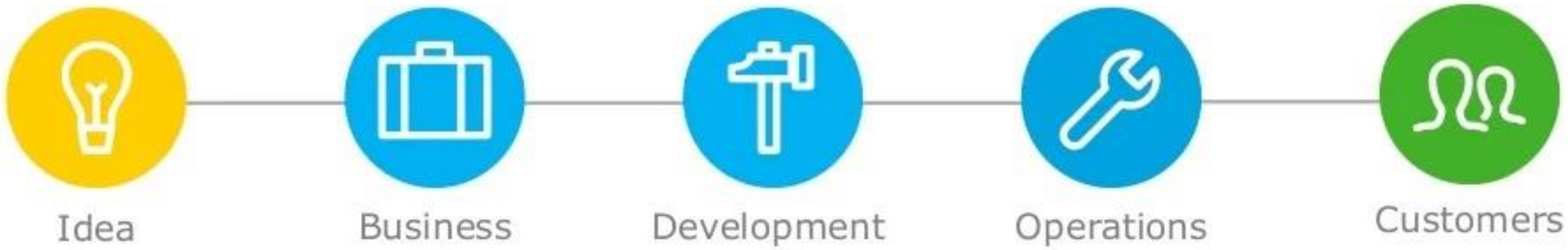
Managing

Technology

Closing

Introduction to Site Reliability Engineering

Typical Software Lifecycle



Sanity Check

The goal is not just for software to be pushed out, but to also be
maintained once it's live

How long does it take for an incident to reach the right team?

Can it be minutes, instead of hours?

How often does the same incident re-occur?

Can it be reduced to *never*?

Reliability is the outcome of a system to be able to withstand some certain types of failure and still remain functional from a user’s perspective.

Site Reliability Engineering…

…is what happens when a software engineer is put to solve operations problems

Site reliability engineering (SRE) is a specialized skillset where software engineers develop and contribute to the ongoing operation of systems in production.

Site Reliability Engineer

Spend 50% of contributions in Systems Engineering problems

--

infrastructure,

Spend 50% of contributions in Software Engineering problems

--

application design, data

SRE team objectives

Support systems in production

Contribute to Project work

SRE approach to Operations:

- Treat operations like a software engineering problem
- Automate tasks normally done by system administers (such as deployments)
- Design more reliable and operable service architectures from the ground-up, not an afterthought

Why another role/discipline?

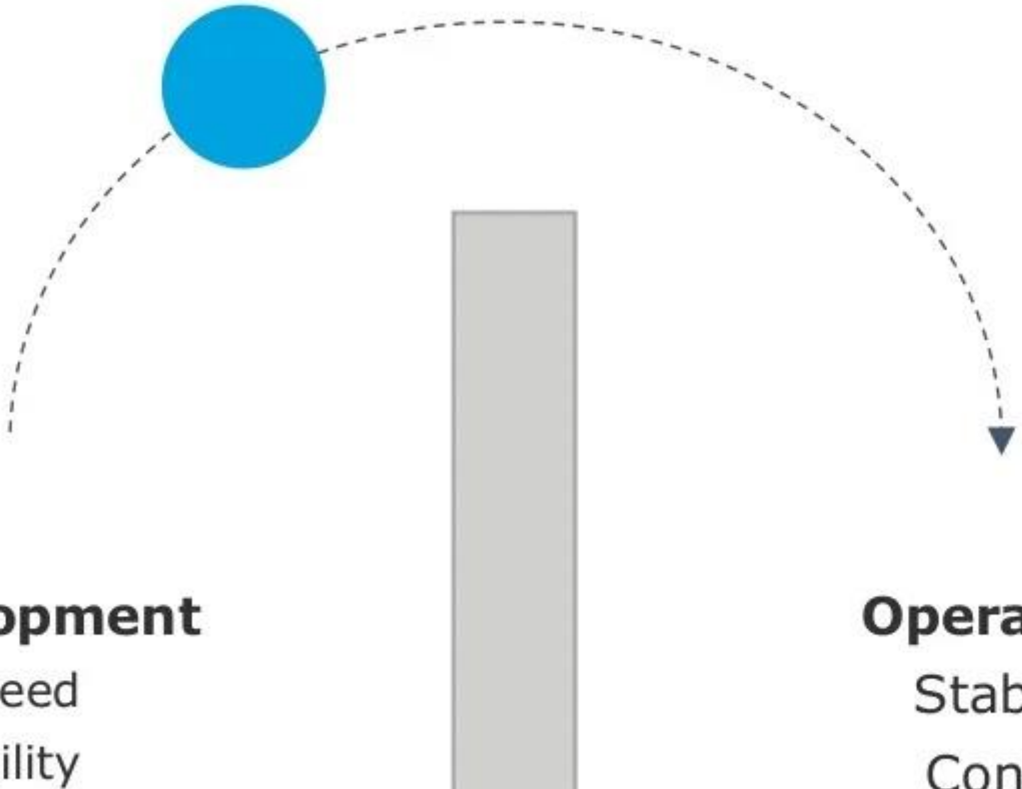
The evolution of systems requires an evolution of systems engineers

Common issues:

- Reliability is usually left to the architects to design, typically at the beginning of a product design
- Non-functional requirements are not reviewed as often
- Change of functionality may impact previously assumed reliability requirements
- Most outages are typically caused by change, and a typical Ops mindset does not necessarily prepare for fast remedy

Wall of confusion

Incentives aren't aligned



Development

Speed

Agility

Operations

Stability

Control

SRE **implements** DevOps

DevOps:

A set of principles & culture guidelines that helps to breakdown the silos between development and operations / networking / security

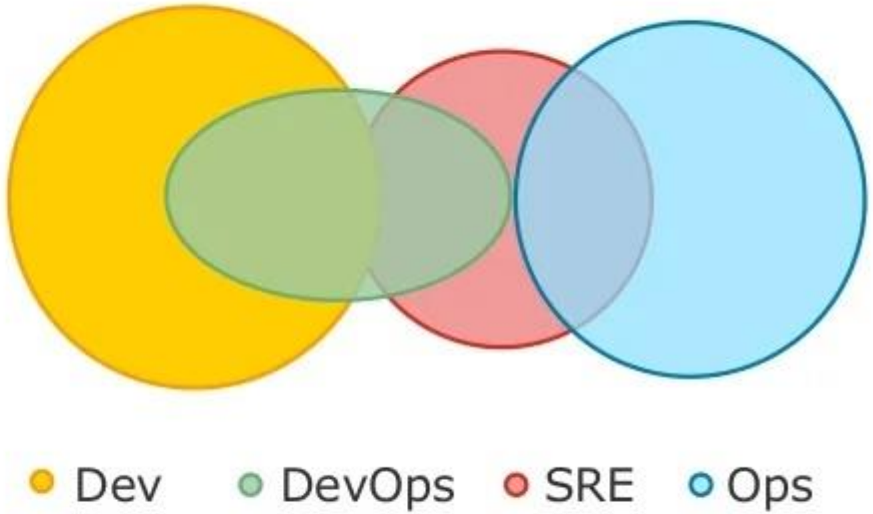
- Reduce silos
- Plan for failure
- Small batch changes
- Tooling & automation
- Measure everything

Site Reliability Engineering:

A set of practices with an emphasis o strong engineering capabilities that *implement* the DevOps practices, an sets a job role + team

- Reduce silos *by shared ownership*
- Plan for failure *using error budget*
- Small batch changes *with focus o stability*
- Tooling & automation *of manual*

SREs aim create more value in linking Operations activities with development



This pattern is used by many organizations that have a high degree of organizational & engineering maturity.

- SRE often act as gate-keepe for production readiness
- Sub-standard software is rejected and SREs provide support in refortifying potential operational issues

Shared Responsibility Model

Application development teams take part in supporting applications in productions

- SRE work can flow into application development teams
- 5% of operational work to developers
- Take part in on-calls (common these days)
- Give the pager to the developer once the reliability objectives are met

Enabling SREs

Designing & setting up SRE in an organization

Rollout of SRE in an Organization

Enabling successful SRE teams requires buy-in across the entire organization

The values of the company must align with how SRE operates

- Organizational mandate
- Leadership buy-in
- Shared Responsibilities

Model 1: Centralized SRE Teams as a shared service

Horizontal enablement of SRE across the organization

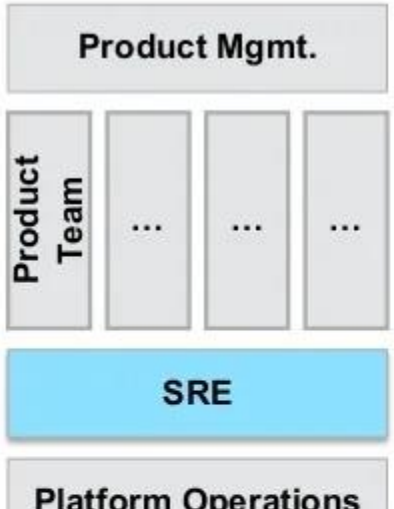
Centralized SRE team that can act as a shared service for all of the organization:

- Common backlog of organization’s reliability improvements – so rolled out evenly
- SREs are brought in as part of design, review, validate, go-lives
- Limited level of influence within team

Sample organizations:

- Facebook (via Production

Centralized teams

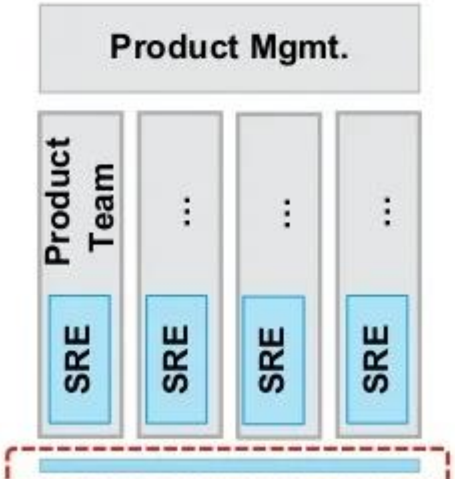


Model 2: Horizontal enablement across products/teams

Placing SREs in vertical slices across the organization

(Embedded with)

Application teams



SREs embedded as full-time team members as part of the product teams:

- Treated like a team member of the product team
- Regular contribution to the overall product reliability design & implementation

Sample organizations:

- Google

Key metrics to measures during enablement of SRE

Using these as a baseline helps to track progress

- Mean Time to Restore (MTTR) service
- Lead time to release (or rollback)
- Improved monitoring to catch & detect issues earlier
- Establishing error budgets to enable budget-based risk management

Hiring & Onboarding SREs

Finding key talent who have natural curiosity for software systems and aggressively automates

Hiring for skillsets:

- Ideally, **multi-skilled** individuals who have experience in writing software ***and*** maintaining software
- **Traditional Ops / Sys Admins** who are willing to learn scripting / programming
- **App developers** who are willing to step up for operational roles
- Exposure to and interest in **systems architecture**

Onboarding SREs:

- **SRE Bootcamps** that consist of insigh into the system architecture, software delivery process and operations
- **On-call rotations** with senior SRE members
- Participation in **incidents response** calls

Budgeting for failures

Error Budgets

Measuring reliability

How do we go about measuring availability of a system *from the eyes of a user?*

$$\text{Availability} = \frac{\text{actual system uptime}}{\text{anticipated system uptime}}$$

- Easy to measure through systematic means (e.g., server uptime)
- Difficult to judge success for distributed systems
- Does not really speak on behalf of the user's experience

Budget for unreliability

Taking one step further, setup a budget that allows for taking risks

100% - Availability target = Budget of unreliability

or, the **Error Budget**

Method:

1. SRE and Product Management jointly define an availability target
2. After that, an error budget is established
3. Monitoring (via uptime) is set in place to measure performance +

Benefits of having Error Budgets

Error budgets become an explicit trackable metric that glues product feature planning to service reliability

Shared responsibility for uptime

Functionality failures as well as infrastructure issues fall into the same error budget

Common incentive for Product teams & SREs

Creates a balance between reliability and feature innovation

Product teams can prioritize capabilities

The team now can decide where and how to spend the error budget

Unavailability Window

Setting up a reliability goal depends on the nature of the business

Availability Target	Allowed Unavailability Window	
	per year	*per 30 days*
95 %	18.25 days	1.5 days
99 %	3.65 days	7.2 hours
99.9 %	8.76 hours	43.2 minutes
99.99 %	52.6 minutes	4.32 minutes
99.999 %	5.26 minutes	25.9 seconds

Approach to defining Service Levels

Product Management with SREs define service levels for the systems as part of product design

Service levels allow us to:

- Balance features being built with operational reliability
- Qualify and quantify the user experience
- Set user expectations for availability and performance

Method to defining Service Levels

Service Level Agreement (SLA)

- A business contract the service provider + the customer
- Loss of service could be related to loss of business
- Typically, an SLA breach would mean some form of compensation to the client

Service Level Objective (SLO)

- A threshold beyond which an improvement of the service is required
- The point at which the users may consider opening up support ticket, the "pain threshold", e.g., YouTube buffering
- Driven by business requirements, not just current performance

Group exercise: Setting up Service Levels

Care-Van is an app that helps to find & book van owners who are available to give lifts to elderly people who have mobility issues.

You are an SRE working with the product team that designs the *ride request for the customers*.

What do you suggest are good measures/metrics of the following:

- Service Level Indicator (SLI)

For Reference

SLI: a measurable attribute of a service that represents availability/performance of the system

SLO: defines how the service should perform, from the perspective of the user (measured via SLI)

SLA: a binding contract to provide a customer some form of compensation if the service did not meet expectations

Sample solution

Service Level Indicator (SLI):

the latency of successful HTTP responses (HTTP 200)

Service Level Objective (SLO):

The latency of 95% of the responses must be less than 200ms

Exhausting an Error Budget

? Question: what can you do if you exhaust the error budget?

Possibly:

- No new feature launches (for a period of time)
- Change the velocity of updates
- Prioritize engineering efforts to reliability

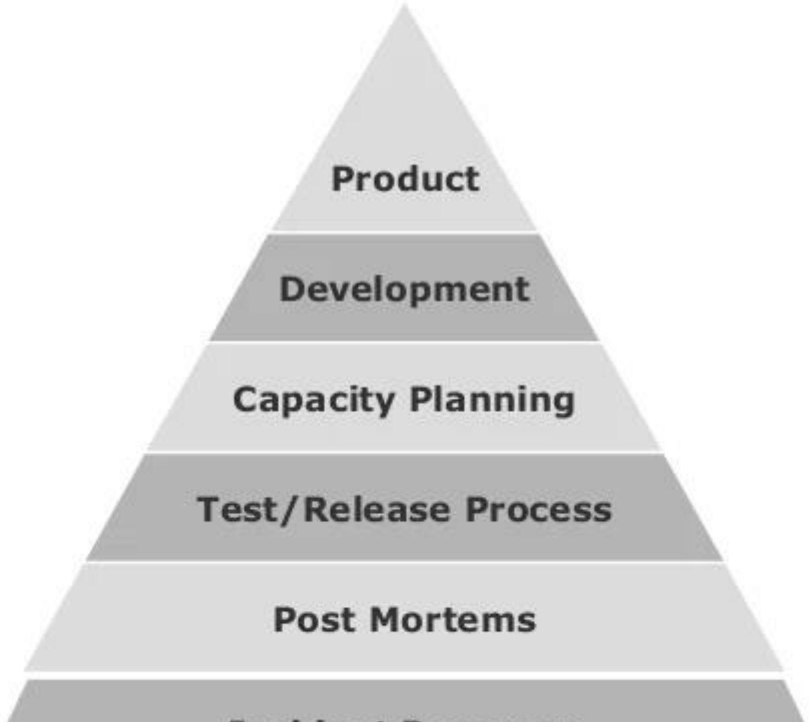
[illegible]

Measuring the Systems

Monitoring & Alerting

Monitoring is a foundational capability for any organization

It is the primary means in determining and maintaining reliability of a system



Monitoring helps with identifying what needs urgent attention, and helps prioritization:

- Urgency
- Trends
- Planning
- Improvements

Golden Signals of Monitoring

The case for better monitoring:

- Focused more on uptime to deduce availability vs. user experience
- Monitoring system metrics, like CPU, memory, storage are good indicators, however may not mean anything meaningful for the client experience
- “Noise overload”

It’s better to other metrics, such as latency: are the response times taking

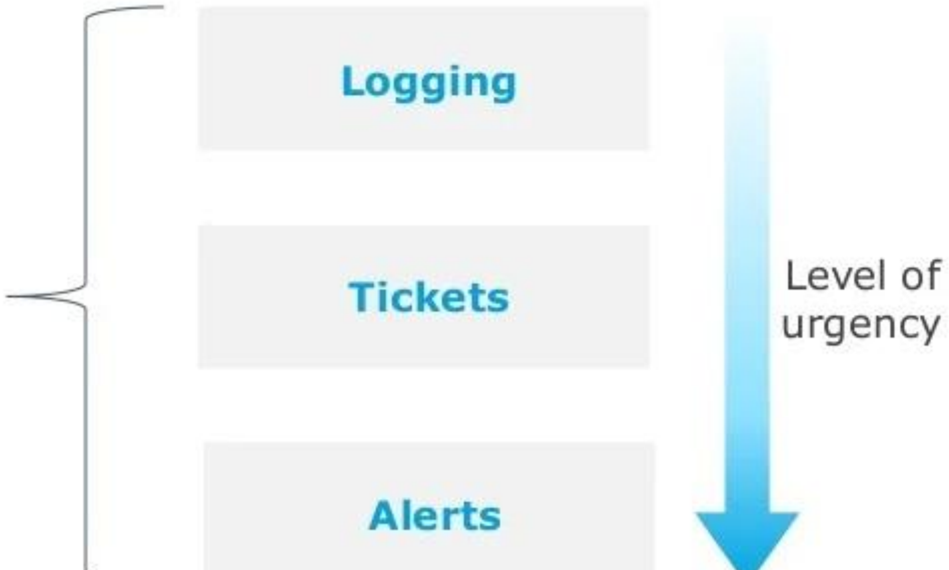
Golden Signals

- Saturation
- Errors
- Traffic
- Latency

✓ It’s best to define SLIs & SLOs using these golden signals as it accounts for

Outputs of a good monitoring system

A carefully crafted **monitoring system** should be able to distinguish urgent requests from important ones, and only alert when required



Alerting design consideration

An alert should only be paged for urgent requests that need immediate attention

Common issues:

- Not all alerts are helpful
- Alerts is not maintained regularly to keep up with product functionality being rolled out

Good practices:

- Measure the system along with all of the parts, but alert only on problems will cause or is causing end-user pain. This helps to reduce "alert fatigue"

Managing incidents

Incident Response & On-Calls

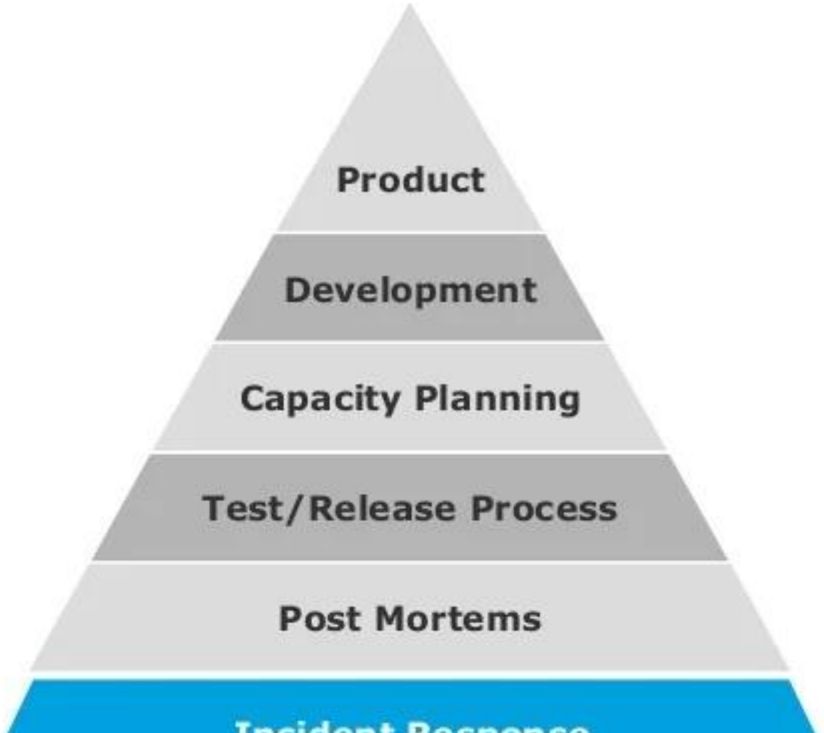
Incident Response

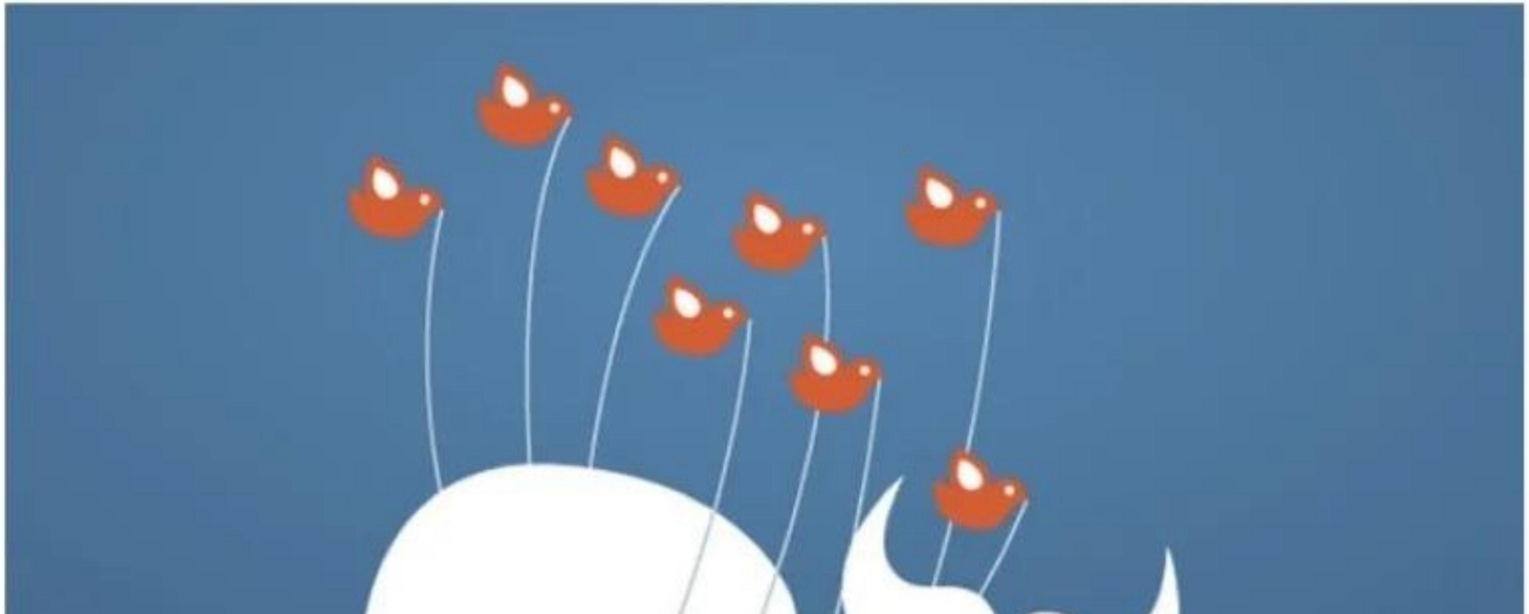
Effective incident response process increases user trust

Users don’t know always expect a service to be perfect.

However, they expect their service provider to be transparent, fast to resolve issues, and learns quickly from mistakes.

Incident Response helps to create that sense of trust when there is a crisis.





Structured Incident Response

People do not typically work well in situations of emergencies, s
having a structured incident response process is essential

1

Have sufficient monitoring

- Service dashboards indicating throughputs, loads, latency, etc.
- Typically, an SLA breach would mean some form of compensation to the client

2

Good alerting & communication process

- Notifying on-call member(s) and having an automated escalation process if primary on-call is not reachable
- Notifying end-users before or as soon as there is degraded experience

Playbook-style plans & tools

Post-mortems

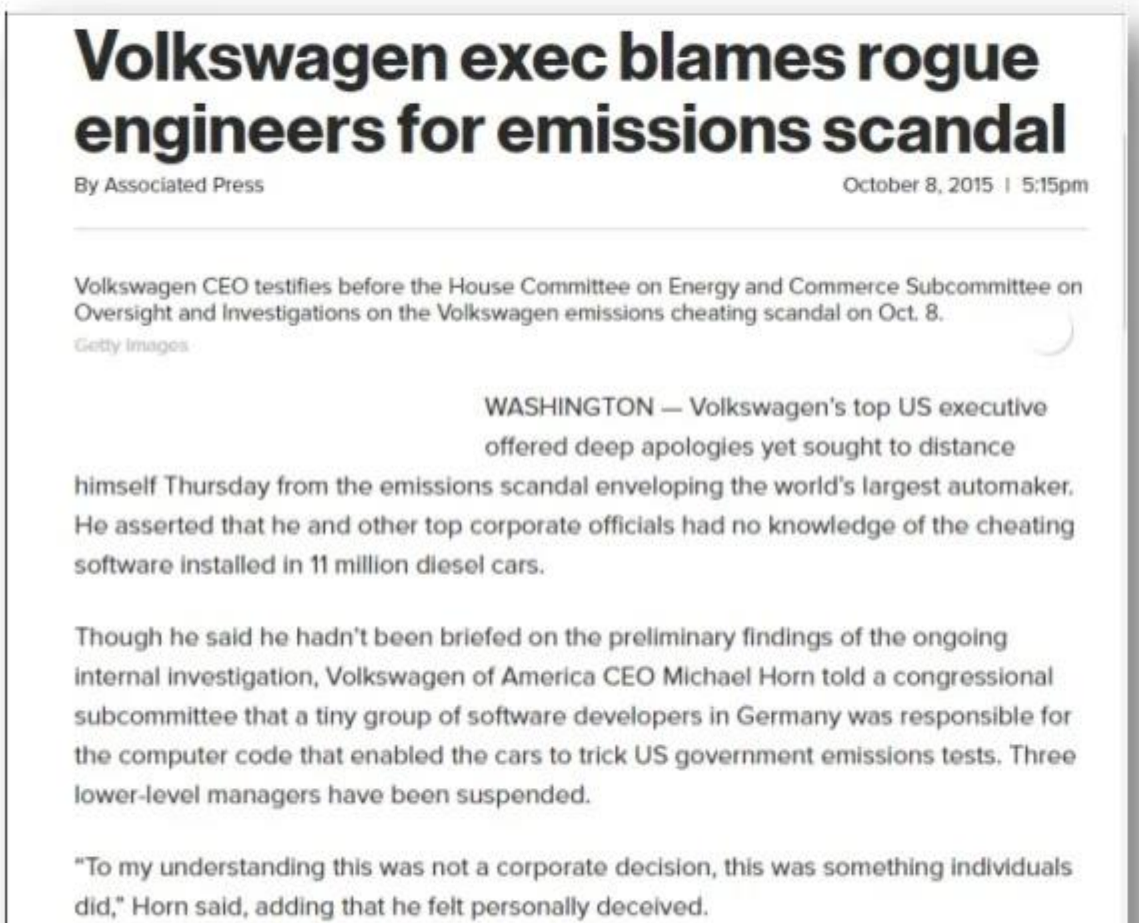
A post-mortem conducted after an incident helps to build a culture of self-improvement

Goals of writing a post-mortem:

- All contributing causes are understood and documented
- Action items are put in place to avoid repeating the same issue again

✓ Good habits:

- Post-mortems conducted & closed within 3-5 days after an incident has occurred
- Document in details errors, logs, steps taken and decisions made that led to the incident/outage



Volkswagen exec blames rogue engineers for emissions scandal

By Associated Press

October 8, 2015 | 5:15pm

Volkswagen CEO testifies before the House Committee on Energy and Commerce Subcommittee on Oversight and Investigations on the Volkswagen emissions cheating scandal on Oct. 8.
Getty Images

WASHINGTON — Volkswagen's top US executive offered deep apologies yet sought to distance himself Thursday from the emissions scandal enveloping the world's largest automaker. He asserted that he and other top corporate officials had no knowledge of the cheating software installed in 11 million diesel cars.

Though he said he hadn't been briefed on the preliminary findings of the ongoing internal investigation, Volkswagen of America CEO Michael Horn told a congressional subcommittee that a tiny group of software developers in Germany was responsible for the computer code that enabled the cars to trick US government emissions tests. Three lower-level managers have been suspended.

"To my understanding this was not a corporate decision, this was something individuals did," Horn said, adding that he felt personally deceived.

VW's defeat device scandal Oct 2015

Conducting **blameless** post-mortem

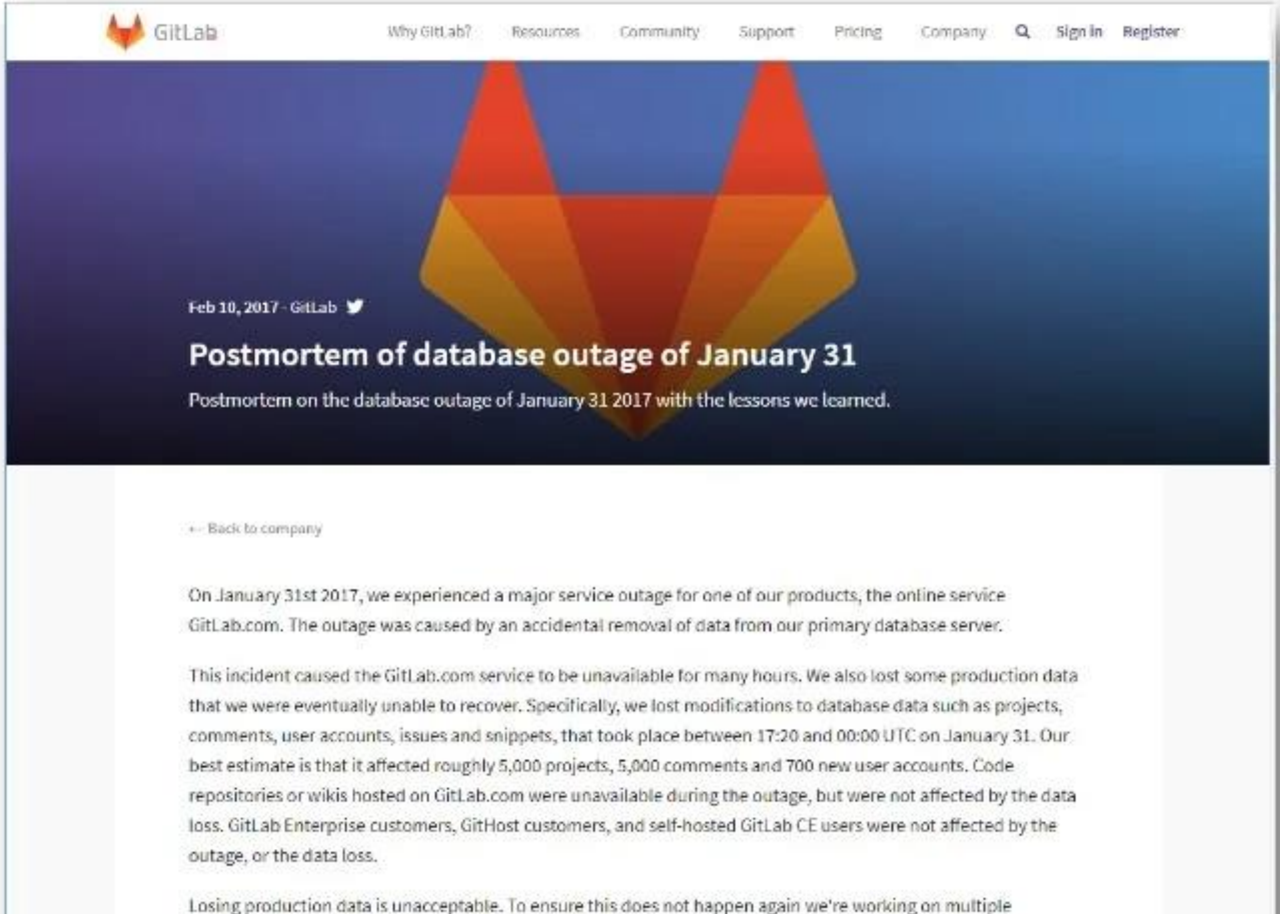
Blameless culture enables faster experimentation without fear. Because failures are part of the process of experimentation.

"Human errors" are system problems. It's better to fix systems & processes to better support people in making good choices.

If the culture of finger-point prevails, there is no incentive for team members to

Useful approach & questions:

- Conduct Five Why's
- How can you provide better information to make decisions?
- Was the information misleading? Can this information be fixed in an automated way?
- Can the activity be automated so it requires minimal human intervention?
- Could a new hire have made the same mistak the same way?



Feb 10, 2017 · GitLab

Postmortem of database outage of January 31

Postmortem on the database outage of January 31 2017 with the lessons we learned.

← Back to company

On January 31st 2017, we experienced a major service outage for one of our products, the online service GitLab.com. The outage was caused by an accidental removal of data from our primary database server.

This incident caused the GitLab.com service to be unavailable for many hours. We also lost some production data that we were eventually unable to recover. Specifically, we lost modifications to database data such as projects, comments, user accounts, issues and snippets, that took place between 17:20 and 00:00 UTC on January 31. Our best estimate is that it affected roughly 5,000 projects, 5,000 comments and 700 new user accounts. Code repositories or wikis hosted on GitLab.com were unavailable during the outage, but were not affected by the data loss. GitLab Enterprise customers, GitHost customers, and self-hosted GitLab CE users were not affected by the outage, or the data loss.

Losing production data is unacceptable. To ensure this does not happen again we're working on multiple

Gitlab's database outage post-morter in January 2017

✓ [illegible]

Better On-Calls

Create a work environments that have a balance between planned/project-driven work and interrupt-driven work

Project-driven work

Planned work to reduce toil

Interrupt-driven work

Supporting incidents

✓ Good practices for creating a more sustainable on-call:

- Have a single person be in charge of on-call (Note: if no one is on call, then everyone becomes on call!)
- On-call person makes sure interrupt-based work is completed without impacting others who are not on-call

Enablers

Technology components

Technology capabilities that help with speed & reliability

Cloud

Cloud architecture is a direct response to the need for agility.

Containers

Infrastructure manages services and software deployments on IT resources where components are

Microservices

Microservices provide a foundation to support automation, rapid deployment, and DevOps infrastructure.

Infrastructure Design

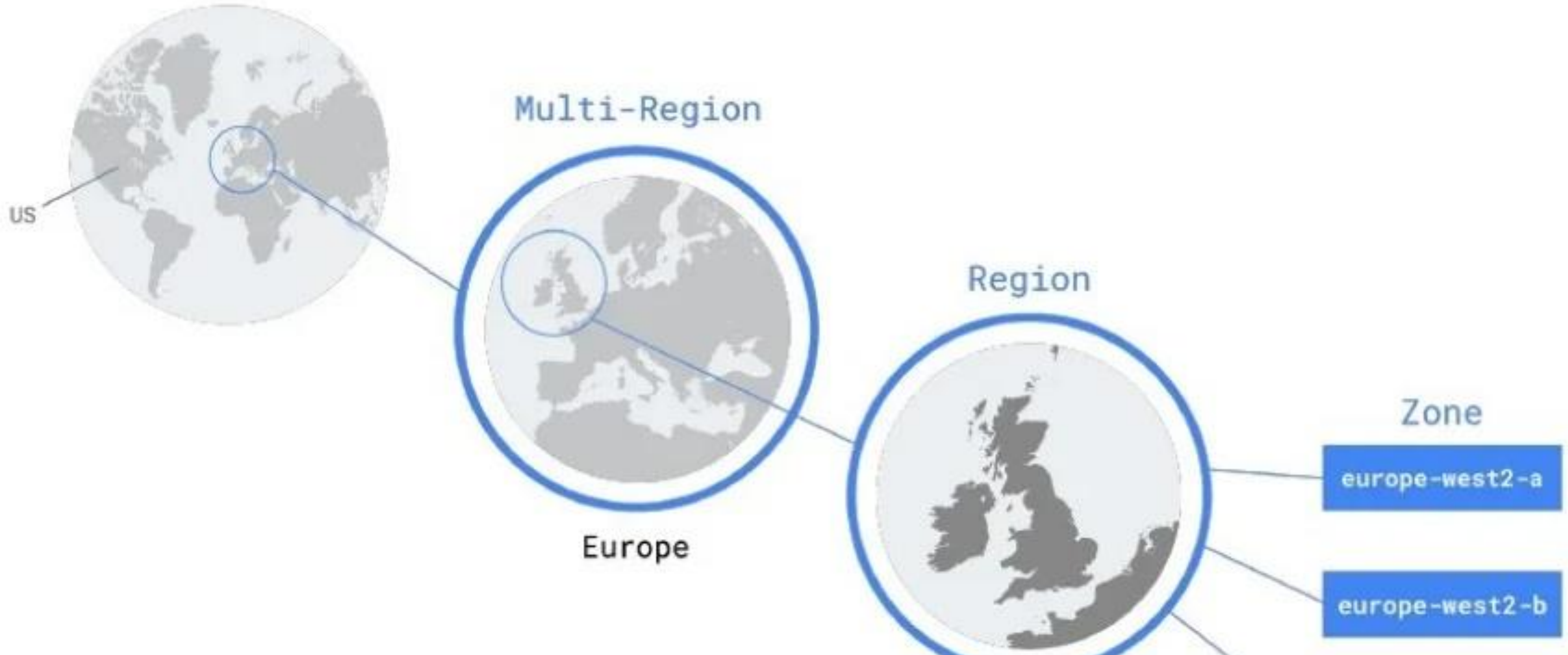
Considerations for storage, compute, networking…

✓ Good practices:

- Version control your infrastructure design and code
- Offload as much work to cloud provider
- Design for reliability by having the application + data situated in more than one datacenter
- Design your infrastructure architecture to support Canary deployment

Using the cloud as an enabler for reliability

Distribute your infrastructure across multiple datacenters (or regions/zones)

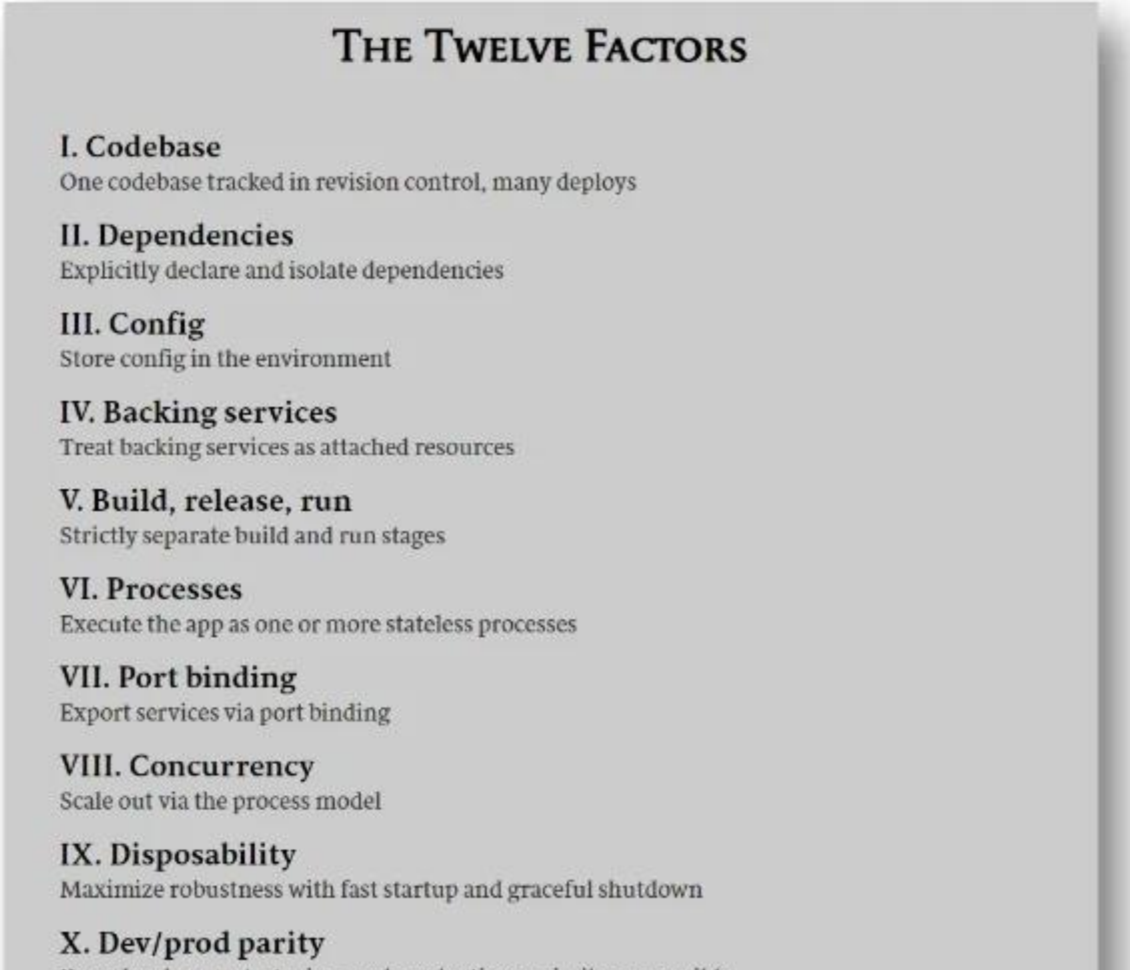


Application design

Considerations for business logic, data layer and presentation…

✓ Good practices:

- Re-think how your features can be re-architected to support for scalability (hence increasing reliability)
- Favour scale out over scale up
- De-couple / modular architecture: such as Microservices
- Distribute your applications across multiple geographies
- Consistent build, deploy process (as much as possible)



THE TWELVE FACTORS

I. Codebase
One codebase tracked in revision control, many deploys

II. Dependencies
Explicitly declare and isolate dependencies

III. Config
Store config in the environment

IV. Backing services
Treat backing services as attached resources

V. Build, release, run
Strictly separate build and run stages

VI. Processes
Execute the app as one or more stateless processes

VII. Port binding
Export services via port binding

VIII. Concurrency
Scale out via the process model

IX. Disposability
Maximize robustness with fast startup and graceful shutdown

X. Dev/prod parity

Design your applications using the principles of Twelve-Factor App

- ✓ Enables "cloud-readiness"

Closing

Recap

Introduction to SRE

Enabling SREs

Budgeting for Failures

Measuring Systems

Get started with your own SRE journey!

Acknowledge: failures are inevitable, it’s best to plan for it

Buy-in: from leadership & top-management

Culture: creating a culture of self improvement

Skillset: look for multi-skilled individuals who strive on collaboration

1. Select a sample application
2. Identify it’s SLI/SLO
3. Create a baseline for current state

Organization-wide Reliability Improvement backlog

Sources of improvement backlog:

- Previous outages and incidents, especially the ones that have been recurrin
- Review of tools used in operations (monitoring, alerting, on-call schedule)
- Operational technical debt – automation of manual scripts
- Mass reboot of all your systems in the whole datacenter
- Do a disaster-recovery (DR) test

Metrics to measures during enablement of SRE

Using these as a baseline helps to track progress…

- Mean Time to Restore (MTTR) service
- Lead time to release (or rollback)
- Improved monitoring to catch & detect issues earlier
- Establishing error budgets to enable budget-based risk management

Metrics to measures during enablement of SRE

...or make one that makes sense to you!

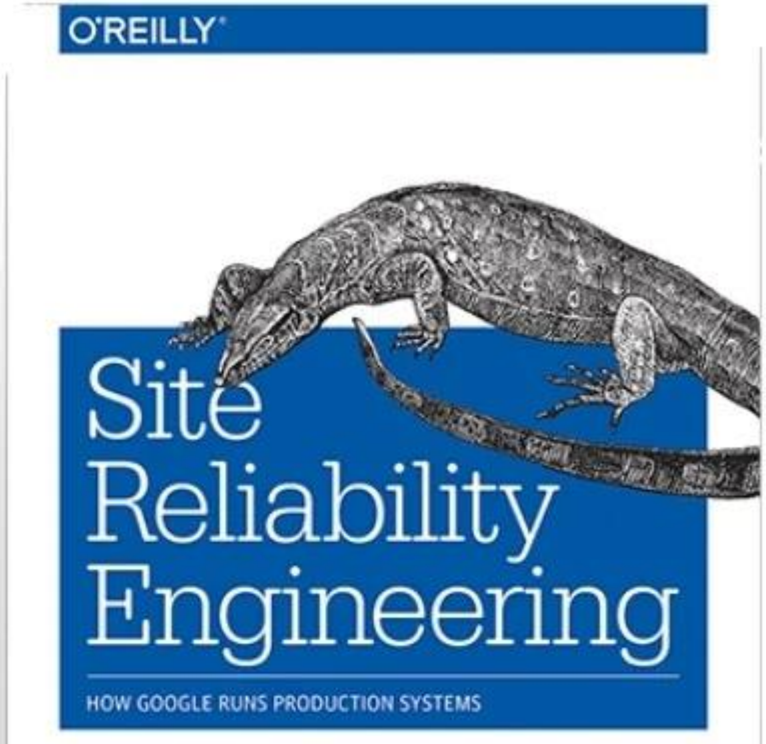
Mean time-to-return-to-bed (MTTRTB)

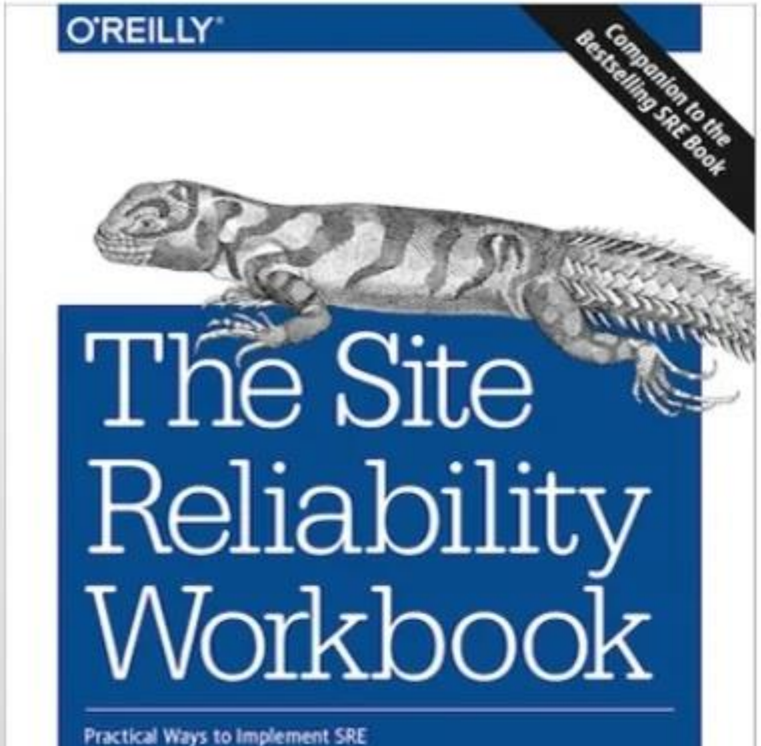
Anti-patterns!

What to watch out for when enabling SRE…

- Rebranding your existing Operations team to SRE is not SRE!
- Treat SREs like any other software developers – not like operations teams
- 100% is not a good SLO. It’s better to plan for failure as so many interactions are out of your hands
- Do not do the transformation in silo. Teams are willing to change; best to involve them as part of the transformation journey. E.g.

Resources


O'REILLY
Site Reliability Engineering
HOW GOOGLE RUNS PRODUCTION SYSTEMS


O'REILLY
Companion to the Bestselling SRE Book
The Site Reliability Workbook
Practical Ways to Implement SRE

Thank you!

Let’s shift operations from **reactive** to **proactive**