A FIELD GUIDE FOR ENGINEERING LEADERS

The Andes Path Playbook

Google's SRE principles, and how we actually use them at Andes Path on real production systems.

WHAT'S INSIDE

ED. 2026 · SANTIAGO / CHICAGO / SAN FRANSISCO

ANDESPATH.COM

SECTION 01

Google's SRE principles: The essentials

Google's Site Reliability Engineering (SRE) treats operations as a software problem. The core truth: reliability is a feature. Engineer it deliberately, or spend the project apologizing for it.

01 **Embrace risk**
Since 100% availability is a fantasy, use error budgets to define and manage acceptable risk.

02 **SLOs as tools**
Service Level Objectives are decision frameworks. Budget remaining means ship fast; budget exhausted means freeze changes.

03 **Eliminate toil**
Systematically remove manual, repetitive work that scales with system size rather than team capability.

04 **Monitor the four golden signals**
Track latency, traffic, errors, and saturation together to gain diagnostic context rather than just noise.

05 **Mature your automation**
Move from manual runbooks toward autonomous self-healing to reduce recovery time and team pressure.

06 **Blameless postmortems**
Focus on learning and root causes rather than attribution, to create the safety needed to surface problems early.

07 **Design for degradation**
Ensure systems fail partially through load shedding and circuit breakers, rather than cascading into total outages.

SECTION 02

The myth of "fixing it later"

Most teams don't fail at SRE because they don't know the principles. They fail because right-sizing is genuinely hard. You are constantly pressured to either over-engineer for a scale you haven't reached, or under-invest until a catastrophe forces your hand.

> "The cost of adopting this from the start is much lower than adding it at the end. By the time you want to be production-ready, if the infrastructure isn't there, you're looking at a massive lift while the team is still expected to ship features."
>
> — CLARK JERIA · CO-FOUNDER, ANDES PATH

Integrated visibility vs. the retrofit tax.

THE TRAP · ADDING IT LATER

Instrumentation layered on after the fact

Alerting is calibrated against "blind" traffic while the team is burning out. You're tuning the system and running it at the same time.

THE ALTERNATIVE · DAY ONE

Four signals wired before the first line of business logic

When you're ready for production, you flip a switch, not a project plan. The data pipeline was always running.

EXAMPLE · CLIENT CONFIDENTIAL

On day one of a robotics project, we built a custom dashboard that displayed battery-temperature data right alongside software error rates. Traditional SRE would keep hardware telemetry and application metrics in separate worlds, but for robotics that separation is artificial: a thermal anomaly in the battery could cascade into software-level failures. Having both on the same pane changed how the team prioritized alerts, because engineers could correlate physical-layer behavior with application errors in real time instead of chasing symptoms in isolation.

SECTION 03

The gap between knowing SRE and practicing it.

Most engineering organizations — and most engineering vendors — fall into one of three categories. Knowing which one you're actually dealing with is the first step.

01

MOST TEAMS

We know the concepts.

SRE is familiar. Engineers have heard of SLOs, golden signals, postmortems. But there's no consistent practice. Observability is spotty. Incidents are handled ad hoc. The knowledge exists but doesn't translate into engineering discipline.

02

BETTER TEAMS

We have some infrastructure.

Dashboards exist. Alerting is configured. Postmortems happen after major incidents, but it was all added after the fact. Coverage is incomplete. Reliability is a parallel workstream rather than a built-in quality.

03

ANDES PATH STANDARD

It's just how we build.

SRE principles are embedded in the technical design process, not added on top of it. Every project ships with the baseline in place. The client decides how much to invest beyond that. The infrastructure to do so is already running.

SECTION 04 · PART 1 OF 4

The automation maturity ladder.

We assess every operational task against this ladder at project kickoff, and build a clear path toward autonomous operation. Most teams never articulate where they sit. We do it before writing a line of code.

L3
TARGET

Autonomous — self-healing.

The system detects issues, isolates root cause, and corrects itself. No humans in the loop. Engineers are notified after resolution, not woken up during it.

```
Detects API saturation
→ auto-scales
→ logs incident
→ notifies team. No page.
```

L2
ACCEPTABLE

Automated — human triggers, machine executes.

Workflows are scripted. Runbooks exist. Humans still initiate remediation, but the steps are codified and repeatable, not improvised under pressure at 2am from a phone.

```
Alert fires
→ engineer runs remediation script
→ documents outcome.
```

L1
MOST TEAMS

Manual — reactive and blind.

Issues surface when users complain. The on-call engineer is the monitoring system. No dashboards, no runbooks, no alerting baseline. Everything is improvised.

```
User emails "the app is broken"
→ engineer logs in and starts
guessing.
```

SECTION 04 · PART 2 OF 4

The four golden signals, as a diagnostic system.

A latency spike without a corresponding error or saturation signal tells you almost nothing useful. All four together show you symptoms for finding the root cause. We instrument all four from day one, whether the client has prioritized reliability work yet or not. The data pipeline is live. The switch is ready to flip.

SIGNAL 01

Latency

How long does it take to serve a request? Track successful and failed requests separately: a fast failure masks completely different problems than a slow success.

CATCHES

slow degradation · timeout storms · third-party dependency failures

SIGNAL 02

Traffic

How much demand is the system handling right now? Traffic spikes often explain latency increases and saturation events before errors appear anywhere in the logs.

CATCHES

unexpected load · viral events · data-pipeline backlogs

SIGNAL 03

Errors

What rate of requests are failing, and how? Track by type. OpenTelemetry can trace an error to the exact function, giving engineers a precise location rather than a search area.

CATCHES

logic failures · integration breaks · gray failures that don't show in latency

SIGNAL 04

Saturation

How full is the system? CPU, memory, connection pools, queue depth. Saturation is often the root cause behind every other signal spike.

CATCHES

resource leaks · under-provisioned infra · cascade-failure precursors

Our rule — you either have 100% signal coverage or you have zero. The moment an engineer has to wonder whether an error was actually captured, the entire system's credibility collapses. The observability layer is mission-critical. It carries a zero-percent error budget.

EXAMPLE · CLIENT CONFIDENTIAL

Our VictoriaMetrics instance went down, which meant we lost visibility into every alert and golden signal we relied on. Even though the robots themselves were still operational, we treated the monitoring outage with the same severity as a full system failure, and we immediately froze all deployments. The freeze lasted about a day until observability was fully restored. Our principle: if you can't see the system, you can't trust the system. Shipping blind is never an option.

Error budgets, as a decision-making tool.

An SLO of 99% availability means a 1% error budget. That budget tells the team how to behave right now, turning a vague notion of "reliability" into an operational framework. We define SLOs at project kickoff so the governance model is running before it's needed.

BUDGET REMAINING

Ship fast.

The error budget exists to absorb calculated risk. When budget is available, teams move aggressively, pushing updates, shipping features, iterating. This is what the budget is *for*.

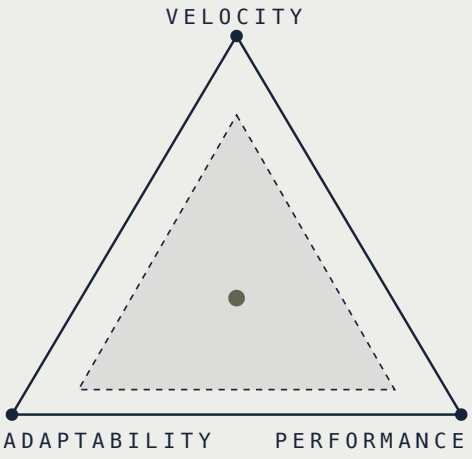
BUDGET EXHAUSTED

Freeze changes.

When the budget runs out, changes stop. Change is the primary source of entropy in software. A stable system that isn't being modified is unlikely to degrade further on its own.

The SRE trade-off triangle.

Every engineering lead is managing a triangle. Velocity, performance, and adaptability pull against each other, and every project has a different right configuration. Understanding which levers you're pulling, and why, is what separates reactive teams from deliberate ones.



VELOCITY

Governed by your error budget.

Budget remaining means you can ship fast and take calculated risk. Budget exhaustion means you stop until stability is restored.

PERFORMANCE

Driven by SLOs, signal coverage, and ladder position.

This is the investment side of SRE: what you buy by instrumenting, automating, and codifying.

ADAPTABILITY

Enabled by reduced toil and codified learnings.

Less manual overhead means faster response to change. Incidents stop producing repeat incidents.

The right configuration depends entirely on context: early prototype, approaching production, or running mission-critical systems. There is no universal setting. Only the right trade-off for where you are right now.

SECTION 04 · PART 4 OF 4

Where Andes Path goes further.

Google's framework is the foundation. We've added to it from years of building production systems, including in robotics, where reliability failures have physical consequences.

01 **Baseline by default**
Every project ships with the four golden signals instrumented and Grafana wired in, regardless of priority. The baseline is non-negotiable. What the client invests beyond it is their call.

02 **Context over dogma**
A prototype and a mission-critical robotics system have different reliability profiles. We calibrate SLOs, automation targets, and investment to your actual context.

03 **Postmortems that create test coverage**
Standard SRE postmortems produce docs and action items. We go further: every incident produces new automated tests. The learning gets codified into the codebase.

04 **Lead ownership, not shared responsibility**
Shared ownership of reliability means diffuse accountability. One named lead owns the SRE posture, makes the trade-off calls, and pushes for the baseline. Everyone else ships.

05 **Lean incident response**
When something breaks: one commander, one comms lead, and only the engineer whose code failed. Not an all-hands war room. The rest of the team keeps building.

06 **Reliability-aware stack selection**
Our rubric for evaluating stacks explicitly includes reliability primitives. We pick frameworks that provide circuit breakers, backoff, and health checks out of the box.

The incident response loop.

Every incident follows this cycle. Step 3 is where Andes Path diverges from standard SRE practice, and where the compounding value comes from.

01 · INCIDENT
Smallest effective team.
Commander, comms lead, subject expert only. Everyone else stays on the roadmap.

02 · POSTMORTEM
Blameless.
What happened, why it happened, what the blast radius was. No attribution.

03 · TEST COVERAGE
Codify the learning.
Every incident leaves the codebase stronger, not just patched. The Andes Path way.

04 · ACTION ITEMS
Fix it, verify it.
Reduce MTTR. Track mean time between failures as a long-term metric.

SECTION 05

The SRE readiness checklist.

30 items across 5 categories. Use it to audit your own team, evaluate a vendor, or give a new engineering lead a concrete starting point. The gaps are where the risk lives.

01 Observability & telemetry

6 ITEMS

- [ ] All four golden signals — latency, traffic, errors, saturation — are instrumented across every service, not just a subset.
- [x] Observability infrastructure was built in from day one, not retrofitted after the system went to production.
- [ ] A visualization platform (e.g. Grafana) is configured and showing live signal data, not scheduled for a future sprint.
- [ ] Error tracking is deep enough to identify the specific service or function causing a failure, not just "API error."
- [x] The observability system itself is monitored and treated as mission-critical; near-zero error budget.
- [x] For hardware or robotics projects, physical-layer errors surface in the same observability pipeline as software errors.

02 SLOs & error budgets

5 ITEMS

- [ ] Service Level Objectives are defined, documented, and reviewed — not implied or informal.
- [ ] Error budgets are tracked on a rolling basis and reviewed regularly, not only after incidents.
- [ ] The team can say *right now* whether it is in "ship fast" or "freeze changes" mode based on the current error budget.
- [ ] Feature vs. reliability trade-offs are made using the error budget as a framework — not gut feel or internal politics.
- [ ] The team is not pursuing 100% availability without a clear, costed business case for the redundancy that requires.

03 Toil reduction & automation

6 ITEMS

- [ ] The team has explicitly identified its top operational time-sinks and ranked them by toil impact.
- [ ] Issue detection is automated; alerts fire before users report problems.
- [ ] Common remediation steps are codified in runbooks, not living only in an on-call engineer's memory.
- [ ] There is a funded plan to move at least one current manual operational process to automated this quarter.
- [x] Customer-facing communications during incidents are templated and sent automatically, not composed under pressure.
- [ ] Mean time to recover (MTTR) is tracked as a formal metric and reviewed after every significant incident.

SECTION 05 · CONTINUED

Design, culture, and scoring.

04 System design & graceful degradation

5 ITEMS

- [ ] The system is designed to serve degraded responses under load rather than fail completely; partial failure is acceptable, total failure is not.
- [ ] Retry logic uses exponential backoff with jitter, not immediate or linear retries that can amplify outages into cascade failures.
- [ ] Load shedding priorities are explicitly defined: which features drop first under heavy load, and which are protected.
- [ ] The technology stack was evaluated partly on whether it provides reliability primitives (circuit breakers, backoff, health checks) out of the box.
- [ ] A technical design review that included reliability and observability happened before the first line of production code.

05 Incident culture & continuous learning

6 ITEMS

- [ ] Every significant incident produces a written, blameless postmortem within 48 hours of resolution.
- [ ] Postmortem learnings are codified into automated tests — not just stored in a wiki page nobody reads.
- [ ] Incident response uses the smallest effective team — commander, comms lead, and the relevant subject expert. No all-hands war rooms by default.
- [ ] Engineers are not the last line of defense. The system alerts the team before users discover an issue.
- [ ] A single named lead owns the team's SRE posture with clear authority to make reliability trade-off decisions.
- [ ] The SRE investment level is explicitly calibrated to the project's current context — prototype, production, or mission-critical — and revisited as context changes.

Your score.

0 – 10
REACTIVE
Your system is a liability waiting to surface. Reliability is the top engineering priority today — not next quarter.

11 – 22
DEVELOPING
Foundation exists but coverage is incomplete. Find the highest-risk gaps and build a funded plan to close them.

23 – 30
PRODUCTION-GRADE
Your team operates with real engineering discipline. The focus now is moving up the automation ladder.

ITEMS MARKED IN OLIVE ARE ANDES PATH EXTENSIONS TO STANDARD GOOGLE SRE PRACTICE.

LET'S BUILD IT RIGHT THE FIRST TIME

You're building the next generation of technology.

We're the team that helps you get it right, and get it built. Andes Path embeds experienced, self-directed engineering pods inside your organization. We don't learn on your project, and we don't leave you with a black box when we're done.

hello@andespath.com
Santiago · Chicago · San Francisco
andespath.com