# Using STPA to Guide Analysis of Platform Services

September 21, 2026

Juan Ignacio Gil, Mercado Libre

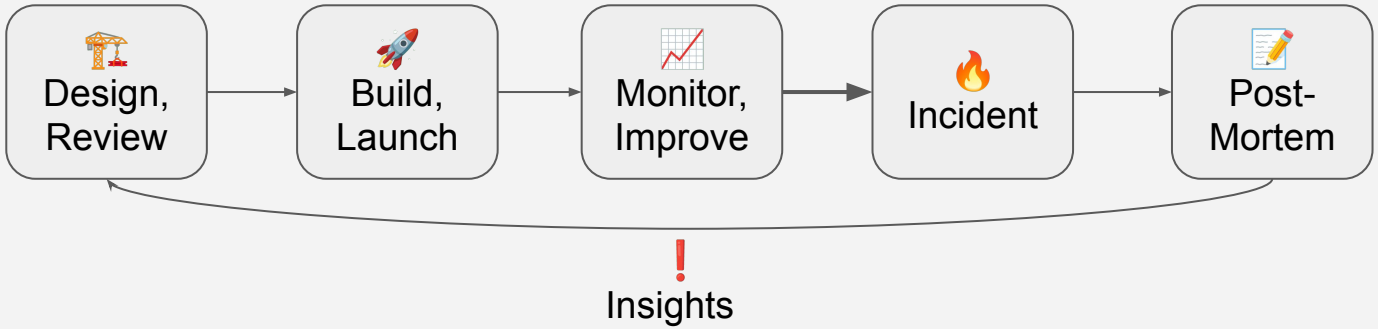
# About me

- Attended 2025 and 2026 STAMP workshops
- Senior Site Reliability Engineer @ Mercado Libre:
  - Biggest e-commerce platform in LATAM
  - ~20,000 developers
  - ~60,000 microservices
  - ~120,000 instances in production

# The team: Services SRE

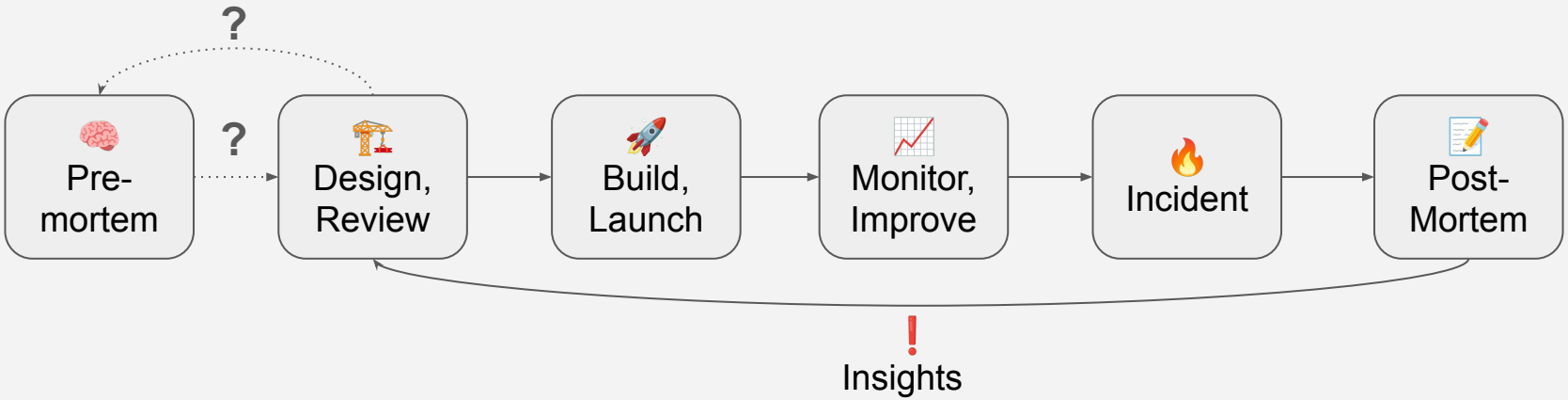
- Main focus: reliability of our internal development platform (Fury)
- Fury services are the building blocks used by developers company-wide
- 9 different types of services: communication, storage, analytics, etc
- Around ~40 different teams
- Small: 6 people
- New: Q4 2025

# Reliability: What's being done?




Incidents kickstart our learning processes -> "Reactive reliability"

# Reliability: What's being done?




Incidents kickstart our learning processes -> "Reactive reliability"

# Reliability: Pre-mortem

For the teams:

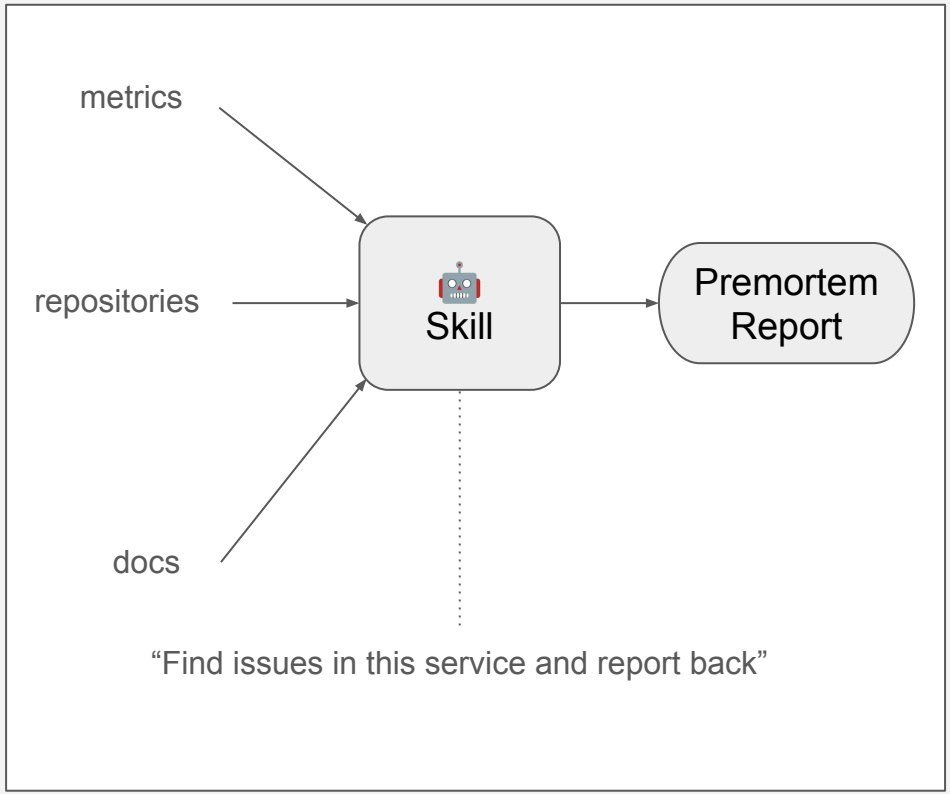
- FMEA-ish document (simpler, SOD + 5 Whys)
- The majority was updated once a year
- Box-checking exercise
- Findings not being followed up on after document revision

# Reliability: Pre-mortem

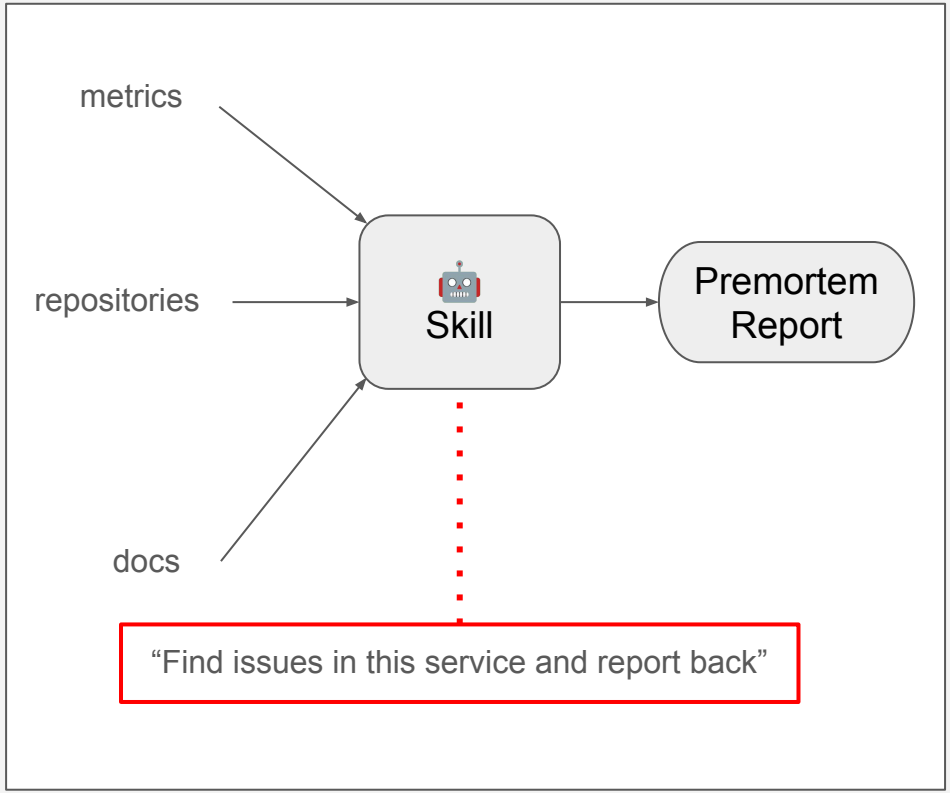
For us:

- Provide value to the existing process
- Opportunity to learn more about the teams
- Build trust
- Possibility to reduce toil

# First version: 2026 ->AI



# First version




Output:

- Hard for us to assess findings
- Findings all over the place: laser focus on code
- A lot of findings: ~20 per service
- Feedback by owners: no real issues

# Change of approach

What if we can abstract what a standard service looks like?

- Services tend to follow similar patterns
- Shared tools (observability, SDKs)
- Most services use a control plane/data plane split

Let’s try STPA!

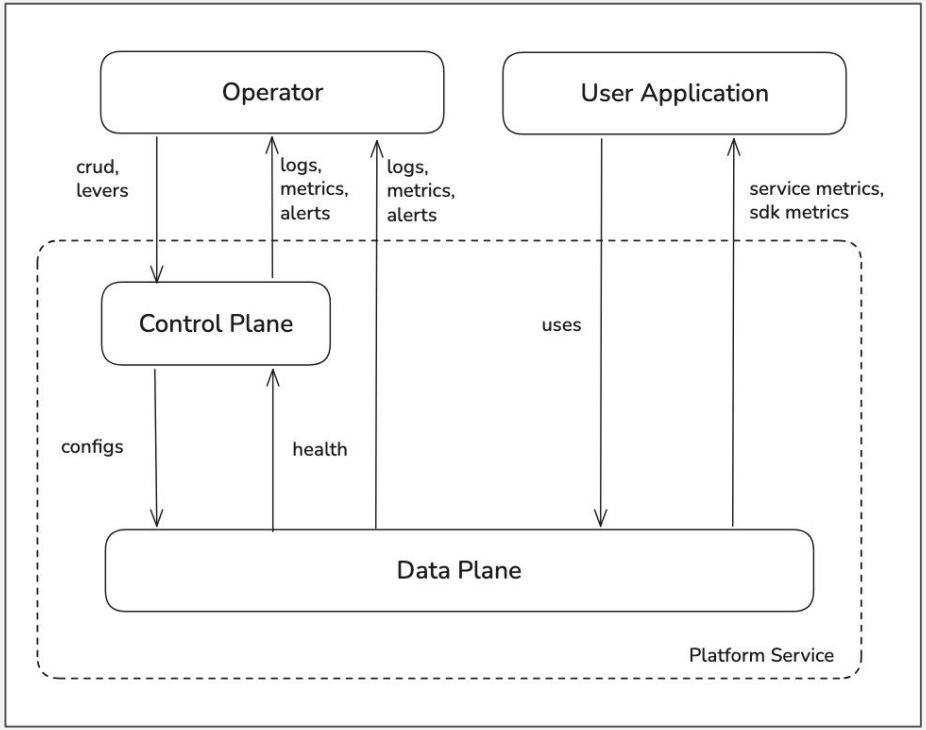
# STPA: Defining losses

- L1: Loss of revenue
- L2: Loss of brand trust
- L3: Loss of platform team trust

# STPA: Defining Hazards

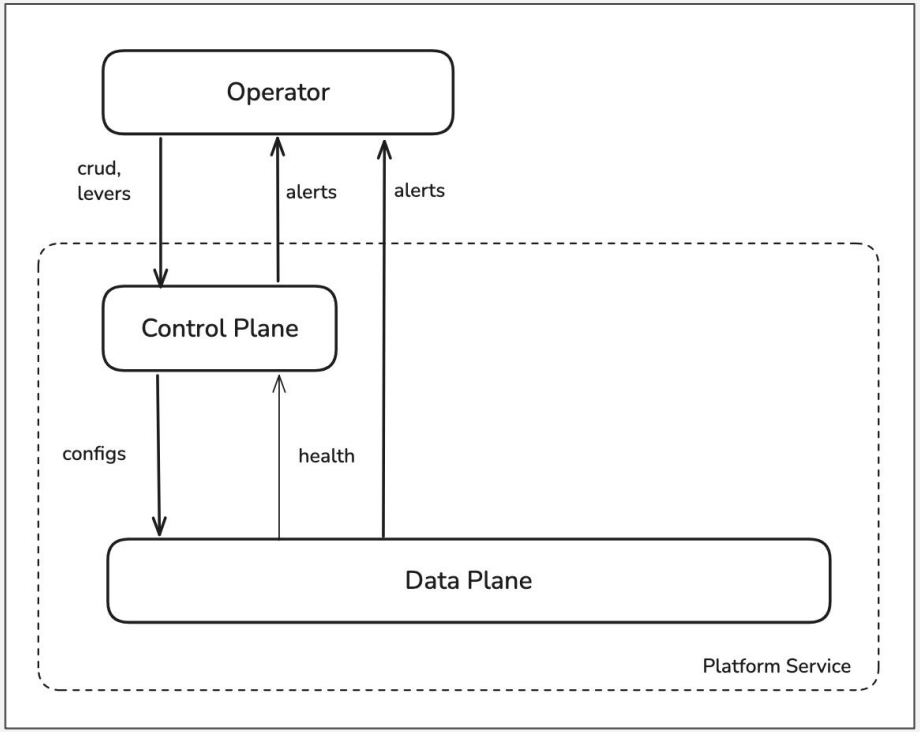
- H1: Service not available
- H2: Service operates with incorrect configuration
- H3: Service allows unsafe deletion

# STPA: Defining a simple canonical control structure




- Represents control plane / data plane interactions clearly
- Operator present in system
- Feedback can be traced to actual channels: log != metric != alert
- Really good companion for postmortem analysis
- UCAs?

# STPA: We stopped before defining UCAs




Why:

- Time
- We want it to be used as input for a model
- It's hard

We had a small pilot test with "Does the data plane stop serving if the control plane becomes unavailable?" and decided to keep the yes/no format.

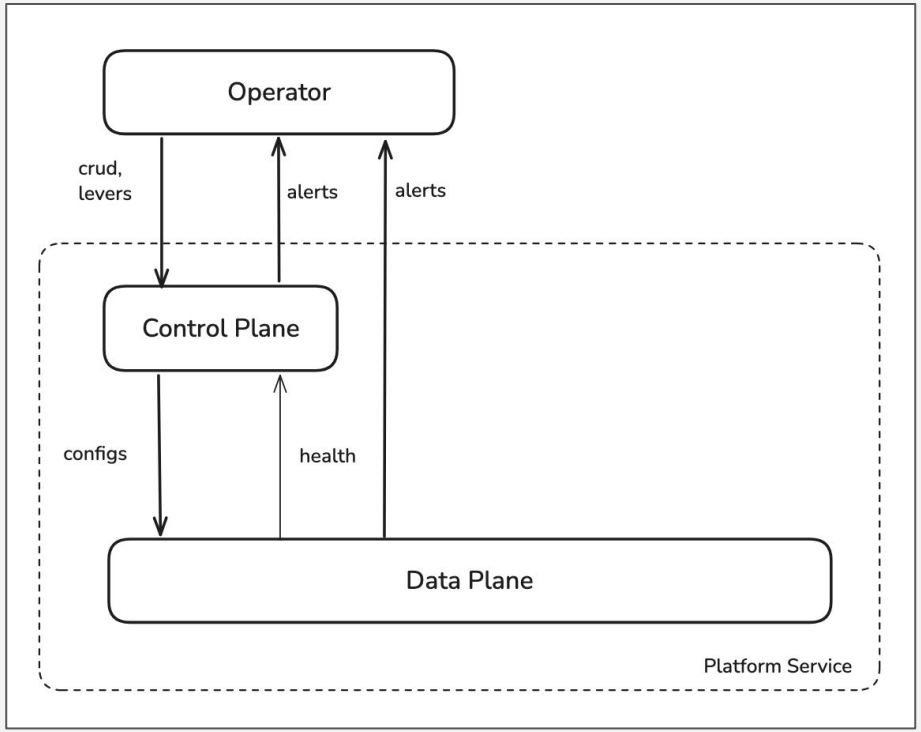
Other checks we defined:

- Are operators alerted if data plane config becomes stale?
- Do new data plane instances fail to boot if control plane is unavailable?
- Can an invalid config be promoted without proper validation?
- Are critical resources deleted without usage checks?
- ..

# Evaluation of the results

- From zero to failure modes took around **a day**
- Findings:
  - **75%** of findings were valid
  - **25%** of them were deemed high/critical by service owners
  - Small to no overlap with existing pre-mortem findings
- STPA:
  - Back-of-the-envelope STPA: powerful and anyone can do it
  - Gateway into systems thinking
- Avoid death by a thousand papercuts:
  - We don't get a single BIG insight; we get to consistently tackle the 'obvious' ones.

# Back-of-the-envelope STPA



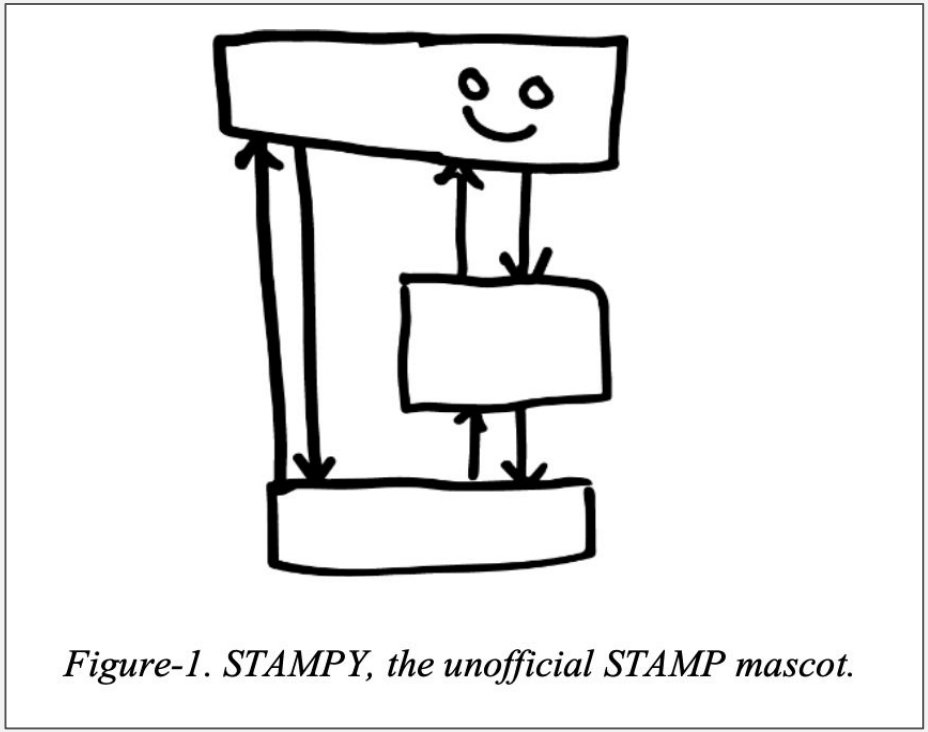



*Figure-1. STAMPY, the unofficial STAMP mascot.*

Source

# Questions?

Related presentations:

STPA Hazard Analysis Practice - John Thomas (MIT); Garrett Holthaus and Theo Klein (Google)

STPA at Meta - Todd Porter (Meta)