O11Y Worst Practices

A measured tier list

Sebastian Schürmann

Principal Architect

MaibornWolff | Less Technology. Better Business.

MaibornWolff

Global Technology Consulting

Less Technology. Better Business.

1000+
Distributed People

4
Countries

2
Continents

75
DC Team Members

Germany • Rwanda • Spain • Tunisia

Our Expertise

- Apps & Cloud
- Cybersecurity
- Data & AI
- Design & UX
- Embedded & Robotics
- Industry 4.0
- IoT
- IT Consulting

Digital & Cloud-native Department

"Someone is always close" - 75 people across 3 countries

Focus: CI/CD • Kubernetes • Infrastructure as Code

Digital & Cloud-native

Department

75
People

3
Countries

"Someone is always close"

Germany • Spain • Tunisia

Our Focus

Everything you call 'Cloud'. We work with and without hyperscalers bringing our expertise in modern cloud-native technologies and practices.

Core Technologies

- CI/CD Pipelines
- Kubernetes & Orchestration
- Infrastructure as Code
- Observability & Monitoring

Project Range

End-to-End (IoT) Development
Gas detection applications with full stack solutions

Cloud Migration & Modernization
Hyperscaler transitions and legacy system updates

Sebastian Schürmann

Principal Architect

About Me

In ‘Software’ for more than 20 years
From Junior Dev to Team Lead: I did all of this with passion
Joined MW 3 years ago
Mentoring others is my thing
At home I have daughter, a large Modular Synthesizer and lots of weird ideas

Germany Digital & Cloud-native

Observability **Cloud-Native**

Experience & Focus

Observability "Expert"

This comes as a side effect from building (cloud native) Apps: You are either good in O11y or you don’t build them

Team Enablement

Helping engineering teams adopt better practices and avoid common pitfalls

Cloud Infrastructure

Kubernetes, CI/CD, Infrastructure as Code, and hyperscaler platforms

Today's Mission

Share the most common observability worst practices I've encountered in the field, ranked by their impact. Learn from our collective mistakes to build better systems.

Tier List Methodology

Definitions

Let's establish a common understanding of key concepts

- O11y
- Worst Practices
- Tier List

O11y

Observability

Definition

The ability to understand the internal state of a system by examining its outputs.

Key Insight

O11y goes beyond monitoring - it's about asking questions you didn't know you needed to ask.

The O11y Flow

Logs **Traces**

Raw events and execution paths

↓

Metrics

Aggregated measurements over time

↓

Alerts

Automated notifications on thresholds

↓

Action

WorstPractices

Anti-patterns

Definition

> **"Still most upvoted on StackOverflow somewhere"**
>
> **"'Heralded' on LinkedIn and HackerNews alike"**
>
> — The persistence of bad advice

Reality Check

Popular doesn't mean correct. These practices spread because they seem to work... until they don't.

Characteristics

Highly upvoted solutions
Popular on developer forums

Widely shared content
Viral on social media platforms

Quick fixes
Seem to solve immediate problems

Hidden consequences
Problems emerge at scale

Copy-paste friendly
Easy to implement without understanding

Learn from our mistakes

TierList

Ranking System

Definition

> **"A tier list is a collection of literally anything you could think of."**
>
> **"A tier list must always be ranked from best to worst."**
>
> — Urban Dictionary

Our Twist

We're ranking worst practices by their potential for career damage and production chaos.

Our Tier System

S — Career-Ending Catastrophes
The elite class of mistakes that can get you fired or make headlines

A — Production Nightmares
Antipatterns that will have you working through the night and canceling weekend plans

B — Technical Debt Black Holes
Problems that slowly consume all available engineering resources with no escape

C — False Sense of Security Specials
Deceptive practices that make you feel safe until disaster strikes

D — Death by a Thousand Alerts
The annoying but persistent issues that slowly drain team morale and attention

Measured by Impact & Pain

The Idea

Why this talk exists

The Format

> **"The Tier List format is good for gatekeeping, a bit preachy and only slightly toxic"**
>
> **"aka great conference content"**
>
> — Self-aware presenter

Slightly Toxic ✓

The Confession

"Then I remembered I had lots of expertise..."

"...in doing things 'not exactly right'"

"aka I made those errors myself (over a long period of time)"

The Real Value

- Learning from collective mistakes
- Shared pain builds empathy
- Humor helps healing
- Prevention through recognition

We've All Been There

O11Y

Worst Practices

A measured tier list of observability anti-patterns, ranked by their potential for chaos

Proceed with Caution


LOG
ALL THE THINGS!
imgflip.com

Data Hoarding

Worst Practice #1

> **"Let's just log everything and figure it out later!"**
>
> — Famous last words of many engineering teams

The Problem

Teams often adopt a "log everything" mentality without considering the downstream consequences. This approach leads to growing storage costs, degraded performance, and ironically makes finding useful information much harder.

Symptoms

- Storage costs exploding
- Query performance degrading
- Signal-to-noise ratio approaching zero
- Time spent searching for relevant data
- Non-compliant by design (ALL regulations)

Critical Impact

Data Hoarding Tier Ranking

Where does "log everything" land on our scale?

S

Career-Ending Catastrophes

The elite class of mistakes that can get you fired or make headlines

Data Hoarding

A

Production Nightmares

Antipatterns that will have you working through the night and canceling weekend plans

B

Technical Debt Black Holes

Problems that slowly consume all available engineering resources with no escape

C

False Sense of Security Specials

Deceptive practices that make you feel safe until disaster strikes

Why S Tier?

- **The Root of All Evil**
  Every other worst practice stems from this
- **Cost Growth**
  Storage costs compound daily
- **Signal Lost in Noise**
  Important events become invisible
- **Query Performance Death**
  Everything becomes unusably slow
- **Security Nightmare**
  PII scattered everywhere

"Just log everything" has ended more budgets than any other single decision in observability.

Remedy: Data Hoarding

Help for "Data Hoarders"

Key Principle

Strategic logging requires intentional decisions about what to capture, how long to retain it, and how to make it actionable.

- Quality over quantity in observability data

Solutions

1. **Consider not logging**
   Question every log entry's value before implementation
2. **Audit and cleanup regularly**
   Regular stack audits to remove unnecessary logging
3. **Configurable severity levels**
   Dynamic log levels without complete rebuilds
4. **Data retention & sampling**
   Intelligent data lifecycle management strategies
5. **PII protection**
   Assume PII cannot be 100% anonymized – but do it anyway – as good as possible

Actionable Solutions


WHEN THE
AUDITOR ASKS IF WE'RE
REDACTING SENSITIVE
DATA IN OUR LOGS
imgflip.com

Security? What Security?

Symptoms

> **"It's just internal logs, who's going to see them?"**
>
> — Famous last words before a security audit

⚠ Reality Check

Internal logs are often the most sensitive data in your organization. They contain everything your application sees, processes, and touches.

Observable Symptoms

GDPR/HIPAA Violations

Personal Identifiable Information (PII) exposed in logs

Names · Emails · Phone Numbers

Credential Leakage

Sensitive authentication data in plain text

API Keys · Passwords · Tokens

Connection String Exposure

Database credentials and internal infrastructure details

DB Passwords · Internal IPs

No Access Controls

Everyone can see everything, no RBAC on log access

Unencrypted Storage

Sensitive logs stored without encryption at rest

Security? What Security? Tier Ranking

Where does security negligence land on our scale?

S

Career-Ending Catastrophes

The elite class of mistakes that can get you fired or make headlines

A

Production Nightmares

Antipatterns that will have you working through the night and canceling weekend plans

Security? What Security? ⚠

B

Technical Debt Black Holes

Problems that slowly consume all available engineering resources with no escape

C

False Sense of Security Specials

Deceptive practices that make you feel safe until disaster strikes

Why A Tier?

Legal Compliance Issues
GDPR fines can reach 4% of annual revenue

Data Breach Incidents
PII exposure leads to mandatory reporting

Credential Compromise
Leaked API keys enable lateral attacks

Emergency Response
Immediate credential rotation required

Reputation Damage
Public disclosure requirements

⚠ **Security incidents create immediate crisis situations requiring all-hands response, but are often not even recoverable with proper incident management.**

Remedy:
Security Issues

Protecting sensitive data

Security First Approach

Observability security requires proactive measures, not reactive fixes. Implement security controls from day one.

Key Principle:
Assume all logs will eventually be compromised.
Design accordingly.

Actionable Solutions

PII Redaction Tools

Available to all developers with known features

Automatic Detection | Real-time Masking

Regular Security Audits

Audit yourself on a regular basis

Automated Scanning | Compliance Checks

RBAC All Log Access

Role-based access control for all observability data

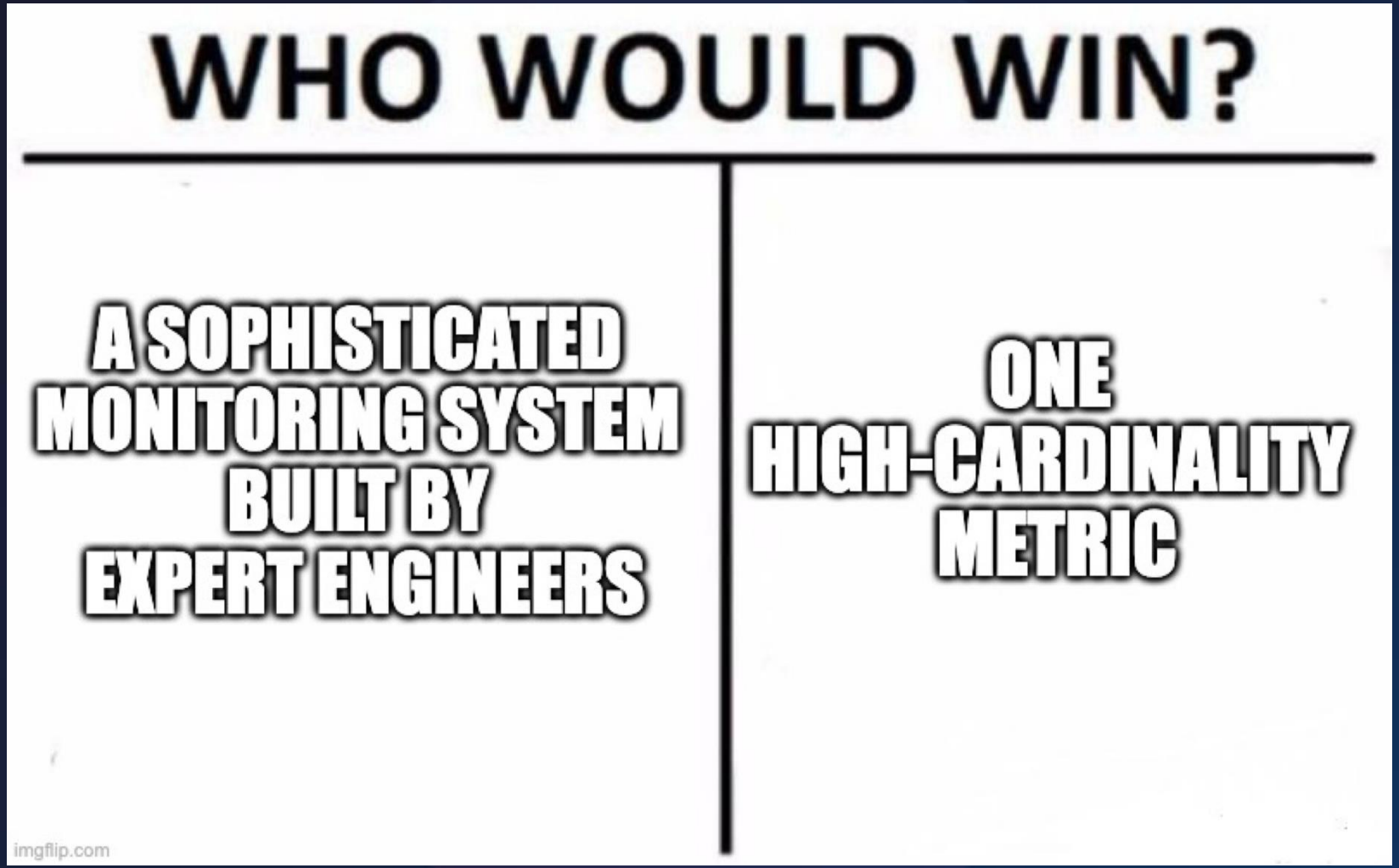
Principle of Least Privilege

Encryption at Rest

Spending money on encryption for data at rest is a safeguard

AES-256 | Key Rotation

Implementation Order:
Start with PII redaction and auditing, then implement RBAC, finally add encryption.


WHO WOULD WIN?
A SOPHISTICATED MONITORING SYSTEM BUILT BY EXPERT ENGINEERS
ONE HIGH-CARDINALITY METRIC
imgflip.com

Runway High-Cardinality

Symptoms

> "Let's just add user_id as a dimension to everything!"
>
> — The moment your metrics database exploded

High-cardinality metrics are like compound interest - they start small but grow exponentially until they consume everything. ⚠

Observable Symptoms

Hard to Predict Log Sizes

Storage costs become completely unpredictable

Exponential Growth Budget Overruns

Query Execution Times Vary Wildly

Same query takes seconds or minutes randomly

Timeouts User Frustration

Excessive Query & Dashboard Hacking

Engineers spend more time optimizing queries than building features

Time Waste Complex Workarounds

Wrong Dimension Dependencies

Other systems inherently using this 'wrong dimension'

Cascade Effects System-wide Impact

Memory & Storage Explosion

Metrics databases consuming entire infrastructure budgets

Runway High-Cardinality Tier Ranking

Where does cardinality explosion land on our scale?

S

Career-Ending Catastrophes

The elite class of mistakes that can get you fired or make headlines

A

Production Nightmares

Antipatterns that will have you working through the night and canceling weekend plans

B

Technical Debt Black Holes

Problems that slowly consume all available engineering resources with no escape

Runway High-Cardinality

C

False Sense of Security Specials

Deceptive practices that make you feel safe until disaster strikes

Why B Tier?

Exponential Resource Consumption
Starts small but grows to consume entire budgets

Engineering Time Sink
Teams spend more time optimizing than building

Compound Effect
Each new dimension multiplies the problem

Workaround Culture
Creates complex hacks and technical debt

No Easy Escape
Removing dimensions breaks existing systems

High-cardinality is the perfect technical debt black hole - it slowly consumes all available resources with no clear escape path.

Remedy: High-Cardinality

Taming the metrics explosion

Prevention > Cure

High-cardinality problems are easier to prevent than fix. Once you have millions of metric series, the damage is already done.

Actionable Solutions

Monitor Query Runtimes & Log Sizes

Track performance metrics to catch cardinality explosions early

Alerting Thresholds | Trend Analysis

Static Code Analysis

Toothfairy: Static code analysis for Query Languages

Pre-commit Hooks | Cardinality Estimation

Backup Analysis Tools

Have an 'anything goes' tool when your O11y solution lacks (e.g. QuickSight)

Emergency Access | Raw Data Analysis

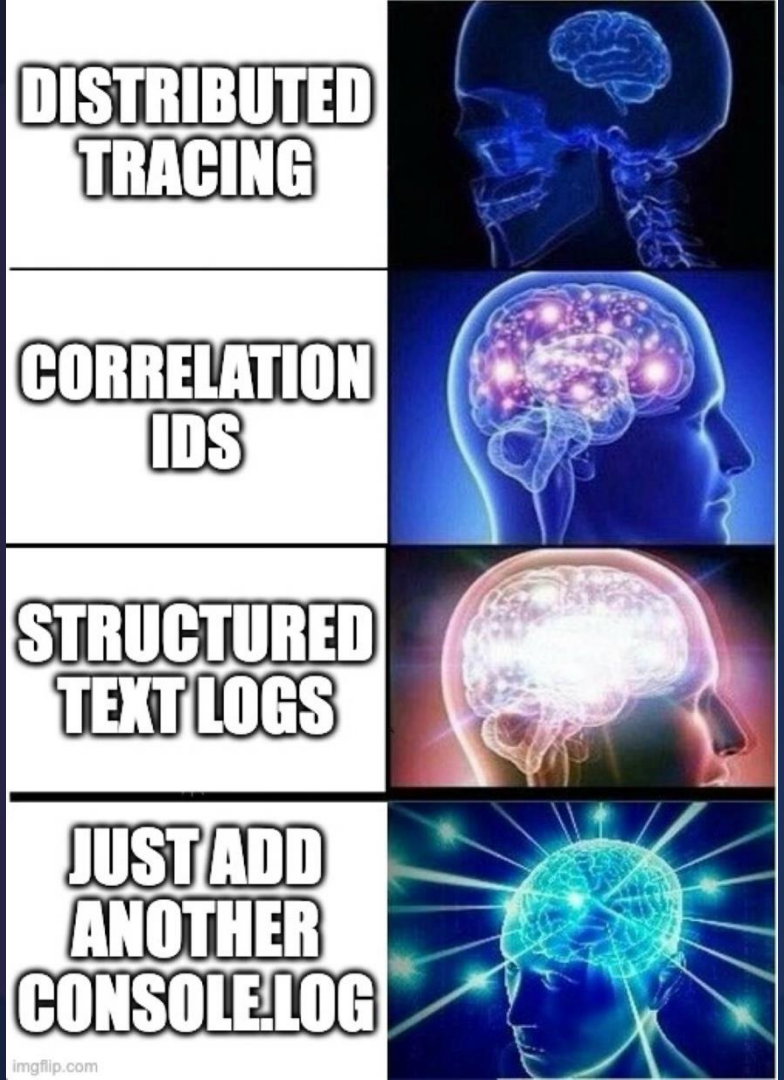
Dimension Governance

Establish approval processes for new high-cardinality dimensions

Review Process | Cost Impact Analysis

Sampling & Aggregation

Use sampling for high-cardinality data and pre-aggregate common queries


DISTRIBUTED TRACING
CORRELATION IDS
STRUCTURED TEXT LOGS
JUST ADD ANOTHER CONSOLE.LOG
imgflip.com

Logs Are Good Enough

Symptoms

> **"Why do we need traces? We have logs!"**
>
> — Famous last words before a production incident

The False(and totally made up) Economy

Traces (avoided cost):	$1,000/month
Extra log storage:	$3,000/month
Debug time overhead:	$5,000/month

Observable Symptoms

Massive Log Volume vs. Minimal Metrics

Terabytes of logs but handful of basic metrics

Storage Costs · Signal vs Noise

Painfully Slow Debugging

Hours spent grep-ing through logs for simple issues

MTTR Explosion · Developer Frustration

Disconnected Service Context

No correlation between services, each log in isolation

Lost Context · Manual Correlation

Sampling Confusion

Inconsistent sampling strategies across different log sources

Data Gaps · Incomplete Picture

False Economy Trap

Trying to save on traces but spending more on logs and debugging time

Logs Are Good Enough Tier Ranking

Where does the logs-only mentality land on our scale?

S

Career-Ending Catastrophes

The elite class of mistakes that can get you fired or make headlines

A

Production Nightmares

Antipatterns that will have you working through the night and canceling weekend plans

B

Technical Debt Black Holes

Problems that slowly consume all available engineering resources with no escape

C

False Sense of Security Specials

Deceptive practices that make you feel safe until disaster strikes

Logs Are Good Enough

Why C Tier?

False Confidence
You think you have observability but you're blind

Hidden Time Bombs
Issues build up until they explode spectacularly

Debugging Nightmare
Hours of manual correlation for simple issues

False Economy
Saves pennies on traces, costs dollars in debugging

Missing Context
No service correlation or request flow visibility

The perfect "False Sense of Security Special" - you feel protected until the moment you need complete observability

Remedy: Logs Mentality

Embracing true observability

The Three Pillars

True observability requires all three pillars working together, not just logs in isolation.

Mindset Shift

Stop thinking "logs are enough" and start thinking "what story do I need to tell?"

Actionable Solutions

Implement Distributed Tracing

Start with OpenTelemetry to trace request flows across services

Request Correlation | Service Dependencies

Build Meaningful Metrics

Focus on business and SLI metrics, not just infrastructure

SLI/SLO Tracking | Business KPIs

Correlation IDs Everywhere

Ensure every log entry can be correlated to traces and user sessions

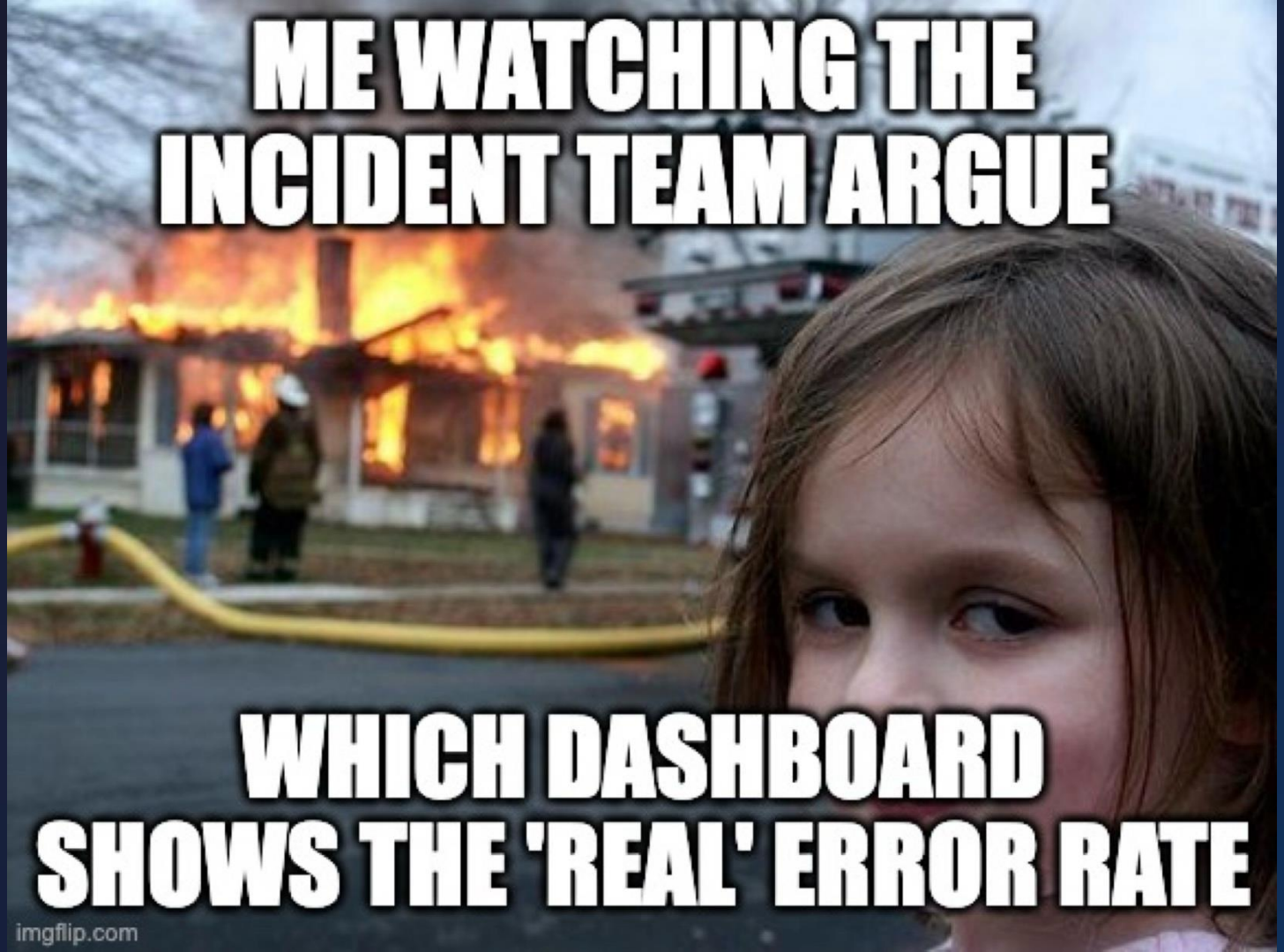
Request Tracking | Cross-Service Context

Smart Sampling Strategy

Implement intelligent sampling that preserves error traces and interesting requests

Error Preservation | Cost Optimization

Team Education


ME WATCHING THE
INCIDENT TEAM ARGUE
WHICH DASHBOARD
SHOWS THE 'REAL' ERROR RATE
imgflip.com

Copy-Paste Dashboarding

Symptoms

> **"Just copy the dashboard from the other team, it should work for us too"**
>
> — The birth of dashboard sprawl

Observable Symptoms

Uniform Looking Dashboards

Every team has the same layout, colors, and chart types regardless of their actual needs

Copy-Paste Culture | No Customization

One Size Fits All Diagram Types

Line charts for everything, even when pie charts or heatmaps would be more appropriate

Wrong Visualization | Missed Insights

Dashboard Wizard Dependency

Only 1-2 people in the organization know how to modify dashboards

Bus Factor = 1 | Knowledge Silos

Point-and-Click Setup

Dashboards created manually through AWS/Azure UI instead of infrastructure as code

No Version Control | Manual Drift

Copy-Paste Dashboarding Tier Ranking

Where does dashboard syndrome land on our scale?

S — **Career-Ending Catastrophes**
The elite class of mistakes that can get you fired or make headlines

A — **Production Nightmares**
Antipatterns that will have you working through the night and canceling weekend plans

B — **Technical Debt Black Holes**
Problems that slowly consume all available engineering resources with no escape

C — **False Sense of Security Specials**
Deceptive practices that make you feel safe until disaster strikes

D — **Death by a Thousand Alerts**
The annoying but persistent issues that slowly drain team morale and attention

Why D Tier?

Slow Energy Drain
Gradually saps team productivity and motivation

Information Overload
Too many similar dashboards, hard to find the right one

Maintenance Burden
Constant requests for "small changes" to copied dashboards

Missed Insights
Wrong visualizations hide important patterns

Team Frustration
Developers lose confidence in observability tools

The perfect "Death by a Thousand Alerts" - not catastrophic, but persistently annoying and draining.

Remedy: Copy-Paste Dashboarding

Treating dashboards as software systems

Core Philosophy

A dashboard displays actionable information: treat it as a software system with proper lifecycle management.

- **Version Control** — Track changes
- **Clear Ownership** — Accountable teams
- **Auto Deploy** — Infrastructure as Code
- **Documentation** — Purpose & usage

Actionable Solutions

Learn Dashboard Frameworks as a Team

Invest in team-wide training on Grafana, CloudWatch, or your chosen platform

Knowledge Sharing | Reduced Bus Factor

Align on Statistical Fundamentals

Ensure team understands when to use different chart types and statistical concepts

Right Visualizations | Better Insights

Infrastructure as Code for Dashboards

Automate dashboard deployment and keep your sanity with version control

Terraform/CDK | Reproducible

Link Dashboards to Documentation

Every dashboard should link to docs explaining its purpose, ownership, and how to act on it

Clear Ownership | Actionable Info

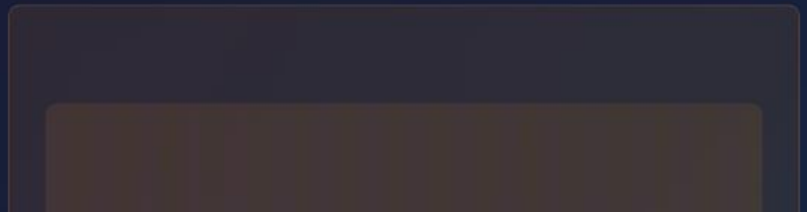
Simple, actionable dashboards

Useless 3D rotating pie charts

Useless, yet Beautiful Wall of Dashboards

Symptoms

> **"Look at our amazing dashboard wall! It shows everything!"**
>
> — Famous last words before an incident



Observable Symptoms

TV Wall Syndrome

Multiple large screens showing dozens of dashboards that nobody actively monitors

Passive Monitoring Information Overload

Alert Blindness

So many red indicators that teams become numb to actual problems

Desensitization Missed Incidents

Decision Paralysis

Teams spend more time deciding which dashboard to look at than actually solving problems

Analysis Paralysis Delayed Response

Vanity Metrics Galore

Impressive looking charts that don't correlate with actual business or system health

Meaningless Data False Confidence

Wall of Dashboards Tier Ranking

Where does the beautiful wall land on our scale?

S — **Career-Ending Catastrophes**
The elite class of mistakes that can get you fired or make headlines

A — **Production Nightmares**
Antipatterns that will have you working through the night and canceling weekend plans

B — **Technical Debt Black Holes**
Problems that slowly consume all available engineering resources with no escape

C — **False Sense of Security Specials**
Deceptive practices that make you feel safe until disaster strikes

Wall of Dashboards

Death by a Thousand Alerts

Why C Tier?

- **False Security Mask**
  Looks comprehensive and professional, hiding the lack of actionable insights
- **Alert Blindness**
  Teams become desensitized to real problems among the noise
- **Delayed Incident Response**
  Critical issues get lost in the visual chaos
- **Cognitive Overload**
  Mental fatigue from processing too much irrelevant information
- **Bystander Effect**
  Everyone assumes someone else is monitoring

The perfect "False Sense of Security" - looks impressive until you need to actually find and fix a problem.

Remedy: Wall of Dashboards

Focus over quantity

Core Philosophy

Less is more: A single focused dashboard that answers "What should I do right now?" beats a wall of beautiful but useless charts.

- **Single Purpose** — One clear goal
- **Actionable** — Clear next steps
- **Owned** — Clear responsibility
- **Timely** — Real-time relevance

Actionable Solutions

Dashboard Audit & Cleanup

Remove dashboards that haven't been accessed in 30 days or can't be explained by their owners

Reduce Noise | Clear Purpose

Create Incident Response Dashboards

Design specific dashboards for different incident types with clear escalation paths

Focused Response | Clear Actions

Implement Smart Alerting

Replace passive monitoring with proactive alerts that trigger specific actions

Proactive | Targeted Response

Assign Dashboard Ownership

Every dashboard must have a clear owner responsible for its accuracy and relevance

Accountability | Maintenance

Look
at dashboard
from
time to time

Create
alert and make
it someone
elses Problem

Alert on Everything Approach

Symptoms

> **"Better safe than sorry - let's alert on everything!"**
>
> — The road to alert fatigue hell

The Alert Storm

Observable Symptoms

Notification Spam

Team members receive hundreds of alerts daily, most of which are false positives or non-actionable

Alert Fatigue | Noise Overload

Alert Fatigue & On-Call Burnouot

Teams routinely disable or ignore alerts because they've learned most are meaningless

Learned Helplessness | Missed Critical Issues

Signal vs Noise Problem

Critical alerts get buried among hundreds of low-priority notifications

Lost Critical Signals | Delayed Response

Alert Triage Overhead

Engineers spend significant time sorting through alerts instead of fixing actual problems

Time Waste | Productivity Loss

Alert on Everything Tier Ranking

Where does the notification storm land on our scale?

S — **Career-Ending Catastrophes**
The elite class of mistakes that can get you fired or make headlines

A — **Production Nightmares**
Antipatterns that will have you working through the night and canceling weekend plans

B — **Technical Debt Black Holes**
Problems that slowly consume all available engineering resources with no escape

C — **False Sense of Security Specials**
Deceptive practices that make you feel safe until disaster strikes

D — **Death by a Thousand Alerts**
The annoying but persistent issues that slowly drain team morale and attention

Why D Tier?

Perfect Death by a Thousand Alerts
Literally the definition of this tier - constant noise that drains morale

Gradual Team Burnout
Slowly erodes team effectiveness and job satisfaction

Alert Blindness
Teams become numb to notifications, missing real issues

Productivity Drain
Time wasted on alert triage instead of actual problem-solving

On-Call Fatigue
Constant false alarms lead to sleep deprivation and turnover

The ultimate "Death by a Thousand Alerts" - it's literally in the name! Persistent, draining, but not

Remedy: Alert Everything

Signal over noise

Core Philosophy

An alert should be actionable, urgent, and require immediate human intervention. Everything else is just monitoring data.

Actionable
Clear next step

Urgent
Needs immediate attention

Human Required
Hard to be automated

Business Impact
Affects users/ revenue

Actionable Solutions

Alert Audit & Cleanup

Remove alerts that haven't triggered actionable responses in the last 30 days

Reduce Noise | Focus Signal

Implement Alert Severity Levels

Critical (page immediately), Warning (business hours), Info (dashboard only)

Prioritization | Appropriate Response

Alert Correlation & Grouping

Group related alerts and suppress downstream notifications during incidents

Smart Grouping | Reduced Spam

Create Alert Runbooks

Every alert must link to documentation explaining the specific response steps

Clear Actions | Faster Resolution


THIS IS WHERE I'D PUT
MY INCIDENT RESPONSE PLAN
IF I HAD ONE!
imgflip.com

Pager Duty Roulette

Symptoms

> "Who's on call this week? I have no idea, let me check the rotation..."
>
> — The sound of chaos in motion

When nobody knows who's responsible, everybody assumes somebody else will handle it.

Observable Symptoms

Mystery On-Call Schedule

Nobody knows who's actually on call, schedules are outdated or non-existent

Confusion | Delayed Response

Alert Ping-Pong

Alerts bounce between team members as everyone assumes someone else is handling it

Bystander Effect | Escalation Chaos

Incident Response Delays

Critical issues take longer to resolve due to unclear ownership and escalation paths

Extended Downtime | Customer Impact

Workload Inequality

Some team members get paged constantly while others never get called

Unfair Distribution | Team Resentment

Pager Duty RouletteTier Ranking

Where does the chaos wheel land on our scale?

S

Career-Ending Catastrophes

The elite class of mistakes that can get you fired or make headlines

A

Production Nightmares

Antipatterns that will have you working through the night and canceling weekend plans

Pager Duty Roulette

B

Technical Debt Black Holes

Problems that slowly consume all available engineering resources with no escape

C

False Sense of Security Specials

Deceptive practices that make you feel safe until disaster strikes

Why A Tier?

Production Nightmare Material
Directly causes extended outages and emergency weekend work

Extended Incident Response
Critical issues take hours longer to resolve due to unclear ownership

Alert Ping-Pong
Critical alerts bounce between team members while systems burn

Team Burnout Acceleration
Unfair on-call distribution leads to rapid engineer turnover

Customer Impact
Delayed incident response directly affects user experience and revenue

Classic "Production Nightmare" - turns every incident into an extended outage with team chaos.

Remedy:Pager Roulette

Predictable responsibility

Core Philosophy

On-call should be predictable, fair, and sustainable. Everyone should know exactly who's responsible and what to do when things go wrong.

- **Predictable** — Clear schedule
- **Fair** — Equal distribution
- **Clear Escalation** — Defined backup
- **Sustainable** — Prevents burnout

Actionable Solutions

Implement Formal On-Call Rotation

Use tools to manage schedules and escalations

Clear Ownership | Automated Escalation

Define Clear Escalation Paths

Primary → Secondary → Manager → Director with specific timeouts for each level

No Single Point of Failure | Guaranteed Response

Fair Rotation Distribution

Ensure equal on-call burden across team members with compensation for extra shifts

Team Equity | Burnout Prevention

Create On-Call Runbooks

Document common incident types with step-by-step response procedures

Faster Resolution | Reduced Stress

Our system has 99.999% availability

Our dashboards show all green metrics

We've optimized all backend performance

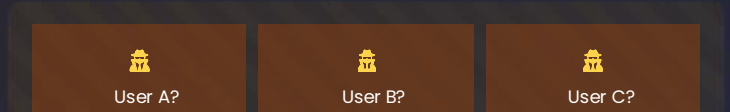
Adblock silently breaks login for 20% of userbase

Users? What Users? Who?

Symptoms

> **"We need 99.99% uptime for everything! Our users demand it!"**
>
> — Someone who has never talked to an actual user

The User Mystery



Observable Symptoms

Cookie-Based Traffic Counting

Teams rely on primitive cookie tracking and basic error counts instead of understanding user journeys

Shallow Metrics

Everything is Equal Importance

All features, all endpoints, all errors treated with the same priority regardless of user impact

No Prioritization | Resource Waste

Excessive Availability Goals

99.99% uptime targets for features that users access once a month

Over-Engineering | Misaligned Priorities

Unknown Exceptions Everywhere

Logs full of "unknown exceptions" and generic error messages with no user context

Meaningless Errors | No Root Cause

Users? What Users? Tier Ranking

Where does user blindness land on our scale?

S

Career-Ending Catastrophes

The elite class of mistakes that can get you fired or make headlines

A

Production Nightmares

Antipatterns that will have you working through the night and canceling weekend plans

B

Technical Debt Black Holes

Problems that slowly consume all available engineering resources with no escape

C

False Sense of Security Specials

Deceptive practices that make you feel safe until disaster strikes

Users? What Users?

Death by a Thousand Alerts

Why C Tier?

False Security Blindness

Teams think they have observability but miss what actually matters to users

Misaligned Priorities

Over-engineering low-impact features while critical user journeys break

Resource Waste

Massive investment in observability that doesn't improve user experience

Product Disconnect

Engineering decisions made in isolation from actual user needs

Delayed Problem Discovery

User issues go unnoticed until they become major problems

Perfect "False Sense of Security" - comprehensive monitoring that completely misses the user

Remedy:
Users What Users

User-centered observability

Core Philosophy

Observability should be built around actual user behavior and business impact, not technical convenience.

Know Your Users
Real behavior data

User Journeys
Critical paths

Business Impact
Revenue correlation

Cross-Team Collaboration
Product + Engineering

Actionable Solutions

Work with Product Management

Establish regular collaboration to understand user priorities and business impact

Aligned Priorities | Business Context

Developers in 1st Level Support

Every developer regularly works support tickets to understand real user problems

User Empathy | Real Problem Awareness

Access Product Analytics

Get direct access to user behavior tracking and product metrics from product teams

User Journey Data | Behavior Insights

Correlate, Don't Duplicate

Stop logging business data - correlate observability with existing database records instead

Reduce Duplication | Single Source of Truth

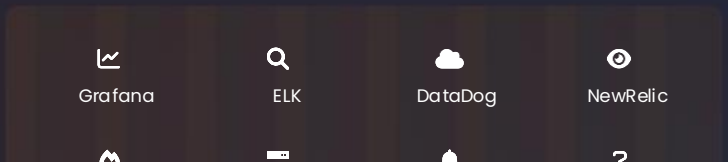

WE SHOULD ALL USE THE SAME OBSERVABILITY PLATFORM
YOUR TOOL DOESN'T WORK FOR OUR USE CASE!
BUT MY TEAM NEEDS SPECIALIZED FEATURES
NOW NOBODY CAN AGREE ON WHAT'S HAPPENING DURING AN OUTAGE!
imgflip.com
WE NEED CONSISTENT DATA ACROSS TEAMS!

Different Teams, Different Tools

Symptoms

> **"We need ONE central platform for all observability!"**
>
> — Management, while teams quietly use 12 different tools

The Tool Fragmentation



Observable Symptoms

Quest for the "One Platform"

Management constantly searches for a single central platform to rule all observability needs

Centralization Fantasy Vendor Shopping

Teams Move Early

Teams adopt different tools independently, driven by immediate needs and personal preferences

Shadow IT Tool Proliferation

Fragmented Standards

Different standards per team, application, platform, or software stack

Inconsistent Practices Knowledge Silos

Integration Challenge

Cross-team incident response becomes impossible due to incompatible tooling

Data Silos Correlation Impossible

Different Teams, Different Tools Tier Ranking

Where does tool fragmentation land on our scale?

S

Career-Ending Catastrophes

The elite class of mistakes that can get you fired or make headlines

A

Production Nightmares

Antipatterns that will have you working through the night and canceling weekend plans

B

Technical Debt Black Holes

Problems that slowly consume all available engineering resources with no escape

Different Teams, Different Tools

C

False Sense of Security Specials

Deceptive practices that make you feel safe until disaster strikes

Death by a Thousand Alerts

Why B Tier?

Perfect Technical Debt Black Hole

Slowly consumes engineering resources with no clear escape path

Exponential Cost Growth

License costs, training overhead, and integration complexity compound over time

Integration Complexity

Each new tool makes the integration problem exponentially harder

Knowledge Fragmentation

Team expertise becomes scattered across incompatible toolsets

Incident Response Delays

Cross-team incidents become coordination nightmares

Classic "Technical Debt Black Hole" - the more tools you add, the harder it becomes to escape the complexity.

Remedy: Different Tools

Pragmatic standardization

Core Philosophy

Accept tool diversity but establish shared standards. Focus on interoperability over uniformity.

Accept Diversity
Tools serve different needs

Shared Standards
Common interfaces

Cost-Driven
ROI-based decisions

Use What Works
Pragmatic over perfect

Actionable Solutions

Accept Different Tooling

Stop fighting tool diversity - different teams have different needs and contexts

Pragmatic Approach | Reduce Friction

Develop Shared Standards

Let teams collaborate on common standards or adopt existing ones (OpenTelemetry, etc.)

Interoperability | Team Collaboration

Use Cost and Effort as Indicators

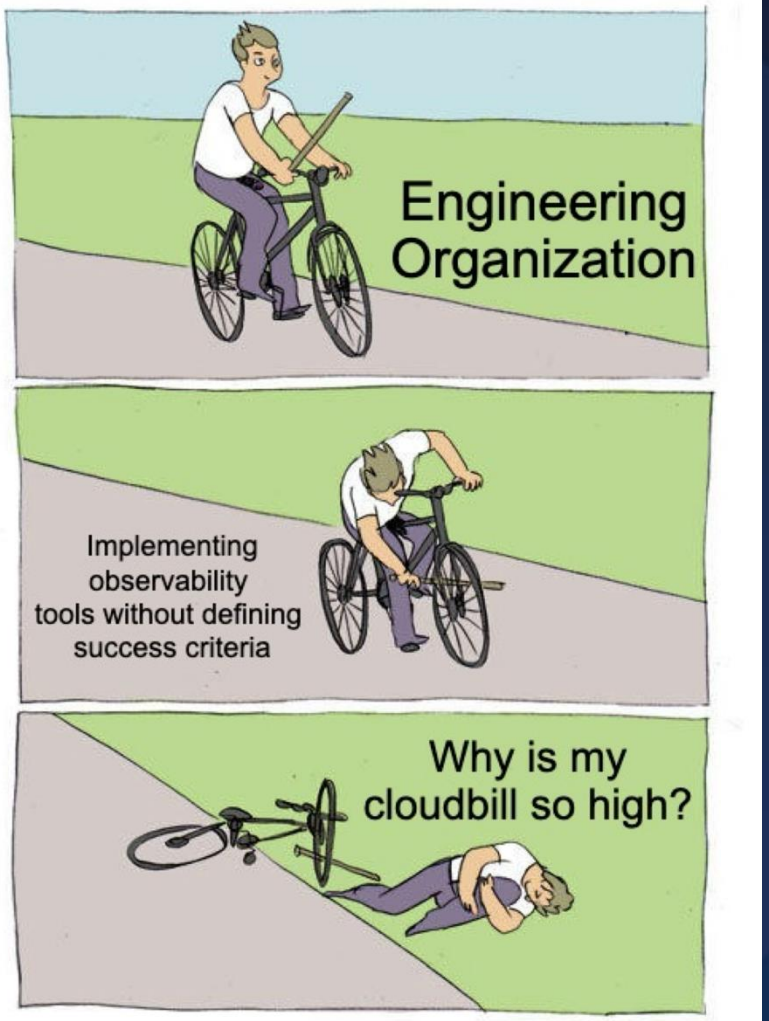
Make tool changes based on ROI analysis, not vendor promises or management mandates

Data-Driven Decisions | ROI Focus

Use What's There Over Hype

Prioritize existing working solutions over shiny new tools that promise everything

Stability Over Novelty | Proven Solutions


Engineering Organization
Implementing observability tools without defining success criteria
Why is my cloudbill so high?
imgflip.com

Missing O11y Objectives

Symptoms

> **"What are our current goals and strategy to reach them?"**
>
> "Uhh... good question..."
>
> — Many teams when asked about objectives

The Objectives Void

? MTTR?	? MTTD?	? RF?

Observable Symptoms

Acronym Confusion

MTTR, MTTD, RF are mysterious concepts that need constant explaining

Knowledge Gaps · No Shared Language

No Quantitative Goals

The question "What are our current goals and strategy?" has no measurable answers

Vague Objectives · No Success Criteria

Product+Dev+Ops Disconnect

Teams work in isolation without shared understanding of business objectives

Siloed Teams · Misaligned Priorities

Baseline Blindness

No understanding of current performance levels or improvement trends

No Benchmarks · Progress Invisible

Missing O11y Objectives Tier Ranking

Where does objective-less observability land on our scale?

S

Career-Ending Catastrophes

The elite class of mistakes that can get you fired or make headlines

A

Production Nightmares

Antipatterns that will have you working through the night and canceling weekend plans

B

Technical Debt Black Holes

Problems that slowly consume all available engineering resources with no escape

C

False Sense of Security Specials

Deceptive practices that make you feel safe until disaster strikes

D

Death by a Thousand Alerts

The annoying but persistent issues that slowly drain team morale and attention

Why D Tier?

Slow Drain on Morale

Teams lose motivation when they can't measure progress or success

Budget Justification Struggles

Constant battles to justify observability investments without clear ROI

Random Walk Development

Teams make decisions without data, leading to inefficient resource allocation

Strategic Misalignment

Engineering efforts disconnected from business objectives

Improvement Invisibility

Cannot demonstrate progress or identify areas needing attention

Classic "Death by a Thousand Alerts" - persistent drain on team effectiveness and strategic clarity.

Remedy: Missing Objectives

Measurable goals drive success

Core Philosophy

Observability without objectives is expensive noise. Set clear, measurable goals aligned with business outcomes.

DORA Metrics
Shared language

Baselines
Know where you are

Business Alignment
CFO partnership

Clear Goals
Measurable outcomes

Actionable Solutions

Set Your Own Objectives

If leadership doesn't provide clear goals, create your own measurable objectives

Take Initiative | Drive Clarity

Start with DORA Metrics

Build shared understanding around Lead Time, Deployment Frequency, MTTR, and Change Failure Rate

Industry Standard | Proven Framework

Calculate Baselines

Establish current performance levels as accurately as possible to measure improvement

Know Starting Point | Track Progress

Align with Business Objectives

Set availability goals that match actual business impact and customer needs

Business Value | Customer Focus

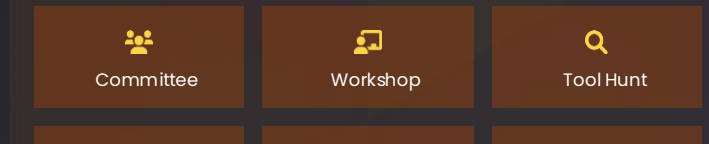

WELCOME TO
OBSERVABILITY THEATRE
imgflip.com

O11yCargocult

Symptoms

> "We need to find THE ONE tool that will solve all our observability problems!"
>
> — Someone who thinks tools are magic

The Cargo Cult Ritual



Observable Symptoms

Committees Stalling Decisions

Endless committees and working groups that discuss observability but never make concrete decisions

Analysis Paralysis | Decision Avoidance

Workshop Theater

Workshops that generate great ideas and beautiful diagrams but zero execution

Ideas Without Action | Planning Theater

Looking for 'The One' Tool

Endless quest for the magical tool that will solve all observability problems

Silver Bullet Syndrome | Tool Obsession

Focus on Tools Instead of Culture

Believing that buying the right tool will magically create observability culture

Technology Over People | Culture Neglect

O11y CargocultTier Ranking

Where does observability theater land on our scale?

S — **Career-Ending Catastrophes**
The elite class of mistakes that can get you fired or make headlines

A — **Production Nightmares**
Antipatterns that will have you working through the night and canceling weekend plans

B — **Technical Debt Black Holes**
Problems that slowly consume all available engineering resources with no escape

C — **False Sense of Security Specials**
Deceptive practices that make you feel safe until disaster strikes

O11y Cargocult

Death by a Thousand Alerts

Why C Tier?

Perfect False Security
Creates illusion of progress while actually preventing insights

Time and Resource Waste
Endless meetings and workshops consume resources without delivering value

Observability Theater
Going through motions without understanding, masking real problems

Tool Obsession
Focus on finding "the one tool" instead of building observability culture

Decision Paralysis
Committees and analysis paralysis prevent any real action

Classic "False Sense of Security" - all the ceremony of observability

Remedy:O11y Cargocult

Action over ceremony

Core Philosophy

Stop talking, start doing. Real observability comes from action and feedback cycles, not committees and workshops.

Start Working
Action beats planning

Feedback Cycles
Learn by doing

Baseline Comparison
Measure progress

Expose Problems
Make issues visible

Actionable Solutions

Get to Work and Create Feedback Cycles

Start implementing observability with "straight thinking" teams who focus on action over process

Action-Oriented | Pragmatic Teams

Compare to Baseline = Win!

Establish current performance baselines and show measurable improvements

Measurable Progress | Concrete Results

Make Deficiencies Visible = Win!

Use observability to expose organizational and architectural problems

Problem Visibility | Systemic Improvement

Skip the Committee Theater

Bypass endless committees and workshops - start with small, action-oriented teams

Avoid Bureaucracy | Direct Action

MaibornWolff
Digital & Cloud-native

Conclusions

O11Y Worst Practices

What we've learned from our journey through observability hell

12 Worst Practices

From career-ending catastrophes to death by a thousand alerts

5 Tier Categories

Gaming-inspired ranking system for maximum impact

The Complete Tier List

Complete O11Y Worst Practices Tier List

S

Career-Ending Catastrophes

- **Data Hoarding** - Storing PII in logs, infinite retention

A

Production Nightmares

- **Security Issues** - PII exposure, no RBAC
- **Pager Duty Roulette** - Unclear on-call ownership

B

Technical Debt Black Holes

- **High Cardinality** - Runway cardinality explosion
- **Different Teams, Different Tools** - Tool fragmentation chaos

C

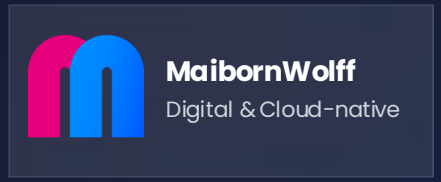
False Sense of Security Specials

- **Logs Mentality** - "Logs are good enough"
- **Wall of Dashboards** - Beautiful but useless
- **Users? What Users?** - User blindness
- **O11y Cargocult** - Observability theater

D

Death by a Thousand Alerts

- **Dashboard Syndrome** - Copy-paste dashboards
- **Alert on Everything** - Notification spam
- **Missing Objectives** - No measurable goals



Questions?

O11Y Worst Practices - Tierlist

Remarks?

Different Opinions?

"The best observability is the one that helps you sleep better at night"

— Sebastian Schürmann, MaibornWolff

sebastian.schuermann@maibornwolff.de