

# Disclaimer

# Disclaimer

The following narrative is a work of pure fiction.

Any similarities to actual people, places, or events are entirely coincidental and absolutely unrelated to the speaker's professional life.

Furthermore, the speaker categorically denies any emotional attachment to animated films featuring entrepreneurial insects.

*This statement has been notarized by the Ministry of Totally Serious Bugs.*





Tathagata Dasgupta (T)



**DEV**

create  
my bugs



**SRE**

ship  
your bugs

# **What Needs to Change Other Than Code**

# You've Patched the Code

- ✓ Hotfix deployed
- ✓ Crisis averted
- ✓ Customer happy

You're only **10% done**

## The Other 90%

The real work—the work that prevents the next fire—happens in the other 90%



# **Three Pillars**

**Measure Baselines**

**Change Workflows**

**Build Partnerships**

# Measure Baselines



If you can't measure, you can't improve

# DORA Metrics as Your North Star

Before changing anything, **measure where you are**  
DevOps Research and Assessment (DORA)

# DORA

Metric	High Performers	Mid Performers	Low Performers
Deployment Frequency	Multiple times/day	Weekly to monthly	Less than monthly
Lead Time for Changes	Hours	Days	Weeks to months
Change Failure Rate	<15%	15-30%	>30%
Time to Restore Service	<1 hour	<1 day	>1 day

# Deployment Frequency

Metric	High Performers	Mid Performers	Low Performers
Deployment Frequency	Multiple times/day	Weekly to monthly	Less than monthly

# Lead Time for Changes

Metric	High Performers	Mid Performers	Low Performers
Lead Time for Changes	Hours	Days	Weeks to months

# Change Failure Rate

Metric	High Performers	Mid Performers	Low Performers
Change Failure Rate	<15%	15-30%	>30%



# Time to Restore Service

Metric	High Performers	Mid Performers	Low Performers
Time to Restore Service	<1 hour	<1 day	>1 day

# The Tech Debt Reality Check

We ship the minimum viable fix because we're always racing to the next fire

# What Accumulates

- **Technical Debt:** *"I'll refactor this later" (never happens)*
- **Process Debt:** Shortcuts that become permanent workflows
- **Knowledge Debt:** Tribal knowledge in one person's head

**Every bug is a loan payment on this debt**

## **2. Process Improvements**

### **The Testing Pyramid Reality**

# Bug Fixes Don't Get the Love They Deserve

✅ New Features: Comprehensive tests

❌ Bug Fixes: Quick patch

**This is backwards.** Every bug fix should add a test that prevents regression.

# The False Security of 100% Coverage

**Unit Tests:** Great for logic, terrible for integration

- Your mocks lie when external service contracts change
- 100% coverage  $\neq$  100% confidence

# Integration Tests: Closer to Reality

But expensive:

- Slow to run, hard to maintain
- Test environments are production's distant cousins

**The Gap:** Further your test environment from production = more bugs slip through

# Testing in Production: The Ultimate Truth

- **Chaos Engineering:** Intentionally break things to find weaknesses
- **Canary Deployments:** Let production tell you what's wrong
- **Feature Flags:** Instant rollback without redeployment

**Game Changer:** DevContainers bring production to your laptop



# Monitoring: When Customers Become Your Alert System

**Red Flag:** If customers report issues before your monitoring does, your monitoring is broken.

# The Bug Story Had:

- Zero disk monitoring
- Zero alerts
- Job silently failed for hours

# Three Monitoring Questions Every Bug Should Ask:

1. **Alert Volume:** Are you drowning teams in noise? (*Alert fatigue kills response*)
2. **Alert Quality:** Do alerts explain the problem AND the next action?
3. **Purpose Clarity:** Dashboards are for debugging. Alerts are for action.

## Remember:

A dashboard is a debugger,  
not an early warning system.

## **3. Tooling Enhancements**

### **The Human Factors**

# Communication in Crisis

The bug story shows coordination chaos:

- Multiple Slack channels
- Different teams, different tools
- Remote, hybrid, multiple timezones
- Stakeholder pressure escalating

# Process Gaps Kill Velocity

Questions your next bug will ask:

- Who owns the support rotation?
- Who verifies weekend releases?
- What happens when the expert is at a conference?
- How do you hand off context across time zones?

# The Locality of Reference Problem

Knowledge lives in silos.

- Person who wrote the code moved teams
- Runbook is outdated
- Tribal knowledge walked out the door

**Solution:** Documentation that lives with the code, not in wikis that rot



# **The AI Question**

**Will Robots Take Our Debugging Jobs?**

## What AI Can Do:

Generate code from prompts

## What AI Can't Do:

- Navigate coordination chaos
- Understand business context
- Make judgment calls under pressure
- Build trust between teams during crisis

# A Bug Is Never Just a Code Change

It's:

- Reproducing the problem reliably
- Understanding system interactions
- Coordinating across teams and timezones
- Making tradeoff decisions under pressure
- Building processes that prevent recurrence

# **The 90% That Matters Most?**

**That's still human work.**

# **The Mindset Shift**

**From Pets to Partners**

# Features vs Bugs: The Adoption Gap

**Features** are planned, nurtured, celebrated—they're **pets**

**Bugs** are unwanted orphans that nobody claims—treated like **pests**

**Truth:** Bugs teach you more about your system than features ever will.

# **Three Organizational Problems Every Team Faces**

# Problem 1: Empathy Deficit

**Solution:** Bugs are inevitable. Budget for them.

- Empathy is bandwidth—measured in time and dollars
- Features add revenue, bugs subtract it
- **Both deserve equal investment**



## Problem 2: Timeline Ownership

**Solution:** Self-empowered teams forecast their own timelines

- Plan backwards from release day
- Map dependencies, approvals, tech debt blockers
- **If you don't plan it, someone unqualified will**

## Problem 3: CI/CD as Assembly Line

**Solution:** Your CI/CD is Ford's conveyor belt—but are you using all its features?


- Team integration via webhooks and APIs
- **If shipping takes more than two weeks, you're waiting too long**

# The Production Gap

Your test environment is production's distant cousin:

- Different data volume, variety, velocity
- Different infrastructure, dependencies, configurations
- **The gap between test and prod is where bugs hide**

## Solution:



Bring production closer to development,  
not the other way around.

# Postmortem Action Items

**WHO** does **WHAT** by **WHEN**

## Short Term

Team	What
SRE	Alert on 85% of disk usage & Runbook update
Data	Error handling, move purger job to Airflow data
UI	Improve Signal to Noise ratio, refactoring third party lib errors
C++	Support rotation calendar update

# Long Term

Team	Focus	DORA Metric
SRE	Faster and reliable updates to production systems	Deployment Frequency
C++	Reduce build times and improve testing efficiency	Lead Time for Changes
Data	Enhance data quality in non-production environments	Change Failure Rate
UI	Implement synthetic UI tests, that will validate post release validations	Time to Restore Service

