Alerting, Runbooks, and Checklists

17-636: **DevOps**

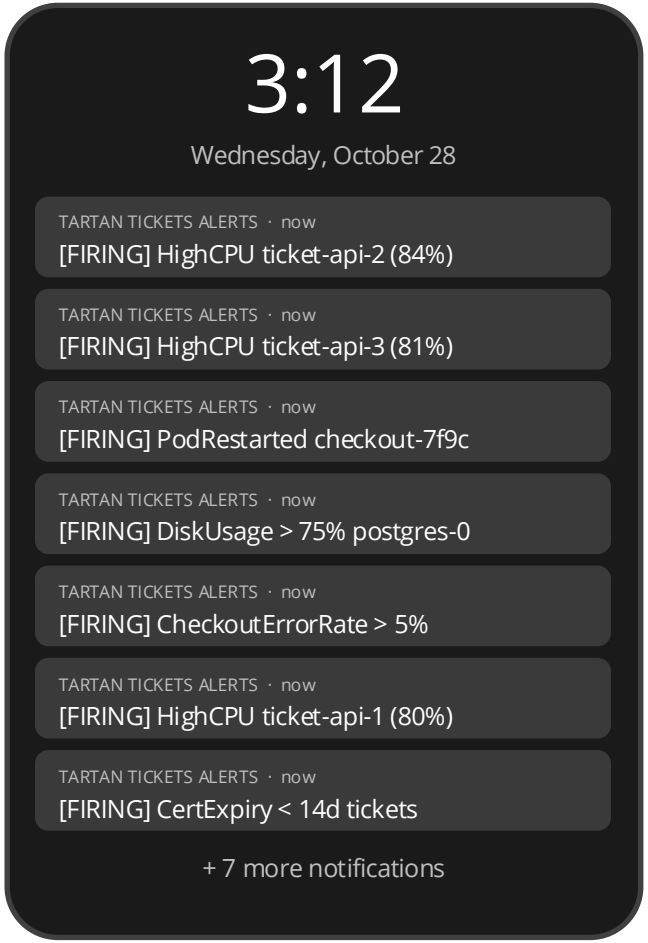
https://cmu-devops.github.io

Michael Hilton

Oct 28, 2026

Learning Goals

- **Distinguish** pages, tickets, and dashboards, and decide which one a signal deserves
- **Explain** how false positives cause alert fatigue, using evidence from hospitals and static analysis
- **Classify** human errors as slips, lapses, mistakes, or violations, and **match** each to a defense
- **Explain** why cognitive load under stress makes "we'll figure it out live" a bad plan
- **Design** a checklist and a runbook that a tired human (or an agent) can follow

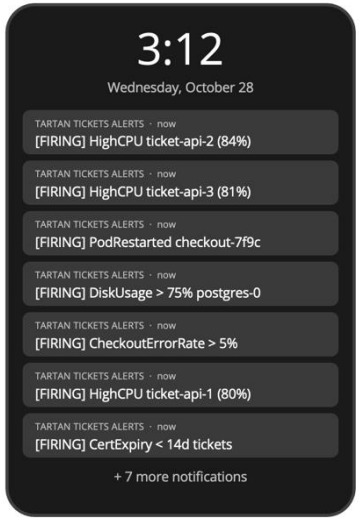

3:12
Wednesday, October 28
TARTAN TICKETS ALERTS · now
[FIRING] HighCPU ticket-api-2 (84%)
TARTAN TICKETS ALERTS · now
[FIRING] HighCPU ticket-api-3 (81%)
TARTAN TICKETS ALERTS · now
[FIRING] PodRestarted checkout-7f9c
TARTAN TICKETS ALERTS · now
[FIRING] DiskUsage > 75% postgres-0
TARTAN TICKETS ALERTS · now
[FIRING] CheckoutErrorRate > 5%
TARTAN TICKETS ALERTS · now
[FIRING] HighCPU ticket-api-1 (80%)
TARTAN TICKETS ALERTS · now
[FIRING] CertExpiry < 14d tickets
+ 7 more notifications

Which one do you look at first?

From Signals to Alerts

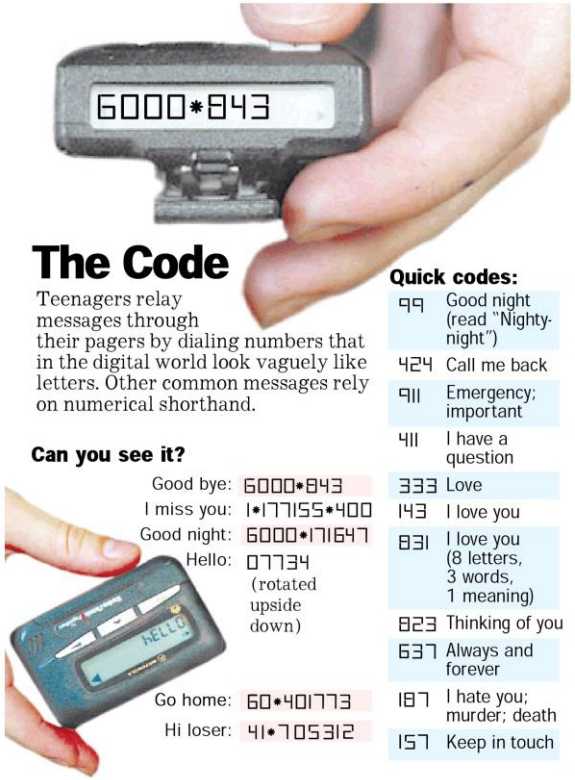
Dashboards Pull, Alerts Push

- **Dashboard:** you go look, when *you* choose
- **Alert:** the system interrupts a human, *now*
- An alert is a claim: **"a human must act, right now"**
- Every alert spends someone's attention, sleep, and trust
- Recall *From Signals to Decisions* (Sep 30): a signal is only useful if it changes a decision





Pagers


6000*843
The Code
Teenagers relay messages through their pagers by dialing numbers that in the digital world look vaguely like letters. Other common messages rely on numerical shorthand.
Can you see it?
Good bye: 6000*843
I miss you: 1*177155*400
Good night: 6000*171647
Hello: 07734 (rotated upside down)
Go home: 60*401773
Hi loser: 41*705312
Quick codes:
99 Good night (read "Nighty-night")
424 Call me back
911 Emergency; important
411 I have a question
333 Love
143 I love you
831 I love you (8 letters, 3 words, 1 meaning)
823 Thinking of you
637 Always and forever
187 I hate you; murder; death
157 Keep in touch
LORENA IÑIGUEZ / Los Angeles Times

Three Destinations for a Signal

Page

- Wake a human **now**
- Users are hurting, or will be within minutes
- e.g., checkout failing

Ticket

- A human should fix it **this week**
- Real problem, not urgent
- e.g., TLS cert expires in 14 days

Dashboard / log

- **No one** is notified
- Useful context when debugging
- e.g., CPU on one pod

Alert on Symptoms, Not Causes

Symptoms (what users feel)

- Checkout error rate > 5%
- p99 purchase latency > 4s
- Zero successful purchases in 15 min

Causes (why it might happen)

- CPU at 85% on one pod
- One container restarted
- Disk 75% full

Page on symptoms. Put causes on the dashboard you open **after** the page.

Beyer et al., Site Reliability Engineering, Ch. 6 "Monitoring Distributed Systems" https://sre.google/sre-book/monitoring-distributed-systems/

Every Page Should Be…

- **Urgent:** it can't wait until morning
- **Actionable:** there is something a human can do
- **Novel:** it needs judgment, not a rote response
  - If a robot could respond, a robot should
- **Real:** users are hurting now, or will be soon

Adapted from Beyer et al., Site Reliability Engineering, Ch. 6 https://sre.google/sre-book/monitoring-distributed-systems/

Every alert spends someone's attention

False Positives and Alert Fatigue

Every Alert Is a Classifier

	Real problem	No problem
Alert fired	**True positive** Caught it	**False positive** Woke you up for nothing
Stayed quiet	**False negative** Users told you first	**True negative** A quiet night

You Can't Make Both Zero

- Move the threshold and you trade one error for the other
- Teams usually react to the **last** incident
  - Missed an outage → lower every threshold → more noise
- Noise has a cost that is easy to ignore, because it's spread out



← Lower threshold: more pages, **more false positives**

Higher threshold: fewer pages, **more false negatives** →

What happens when most pages are noise?

Alarm Fatigue in Hospitals

- Between **72% and 99%** of clinical alarms are false (Sendelbach & Funk, 2013)
- Clinicians respond rationally: turn down volumes, widen limits, silence monitors
- The Joint Commission (2013): **98 alarm-related sentinel events, 80 deaths** (2009–2012)
- Real alarms get lost in the noise



Sendelbach & Funk, "Alarm Fatigue: A Patient Safety Concern," AACN Advanced Critical Care 24(4), 2013. The Joint Commission, Sentinel Event Alert #50, 2013.

Ignoring a noisy alarm is a rational, learned response

Same Problem in Static Analysis

- Static analysis tools find real bugs…
- …and developers routinely ignore them
- Top complaint in interviews: **false positives** and noisy output
- Tools that cry wolf get turned off, and then the real bugs ship



Why Don't Software Developers Use Static Analysis Tools to Find Bugs?

Brittany Johnson, Yoonki Song, and Emerson Murphy-Hill
North Carolina State University
Raleigh, NC, U.S.A.
bijohnso,ysong2@ncsu.edu,emerson@csc.ncsu.edu

Robert Bowdidge
Google
Mountain View, CA, U.S.A.
bowdidge@google.com

***Abstract*—Using static analysis tools for automating code inspections can be beneficial for software engineers. Such tools can make finding bugs, or software defects, faster and cheaper than manual inspections. Despite the benefits of using static analysis tools to find bugs, research suggests that these tools are underused. In this paper, we investigate why developers are not widely using static analysis tools and how current tools could potentially be improved. We conducted interviews with 20 developers and found that although all of our participants felt that use is beneficial, false positives and the way in which the warnings are presented, among other things, are barriers to use. We discuss several implications of these results, such as the need for an interactive mechanism to help developers fix defects.**

I. Introduction

Software quality is becoming more important with the increasing reliance on software systems. There are different ways to ensure quality in software, including code reviews and rigorous testing. Software defects, or bugs, can cost companies significant amounts of money, especially when they lead to software failure [1], [2].

Static analysis tools provide a means for analyzing code without having to run the code, helping ensure higher quality software throughout the development process. There are a variety of ways to perform automatic static analyses [3], including at the developers request, continuously while creating the software in a development environment, and just before the software is committed to a version control system. The tool may allow the developer to configure what kinds of bugs it finds, and sometimes even define new bug patterns. Some automated static analysis software, such as the software integrated into IntelliJ IDEA [4], provide *quick fixes*. A quick fix is a suggested solution for a defect that is automatically applied to a developer's code. To help explain the "state of the art" of static analysis tools, let us look at FindBugs [5] as a concrete example of how these tools work [6]. FindBugs runs as a plug-in for the Eclipse [7] and NetBeans [8] integrated development environments (IDEs). It can also be run from the command line or as a separate tool on its own. When run in the IDE, FindBugs has its own perspective where the defects are listed and organized. Each defect is assigned a severity, signifying how important the defect is; either *high*, *medium* or *low*, each represented by red, yellow and green bug markers respectively. FindBugs offers a select few quick fixes.

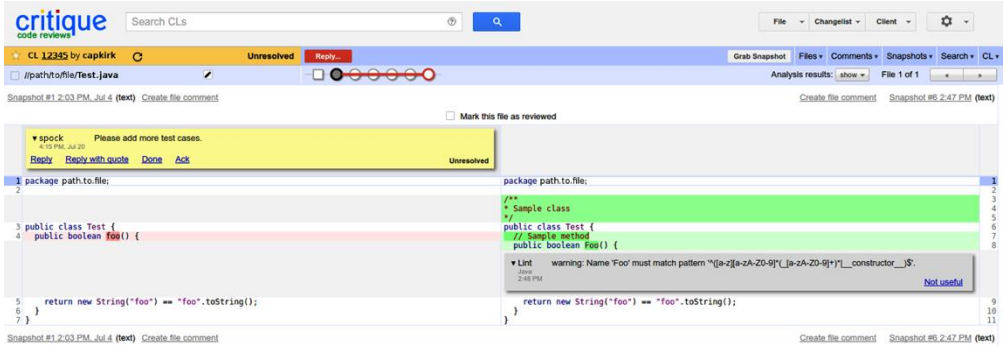
There are many situations where a developer may consider using a static analysis tool to find defects in their code. Let us consider a developer, Susie. Susie is a software developer at a small company. She wants to make sure that she is following the company's standards while maintaining quality code. She needs a way of checking her code in her IDE, before submitting it to the general code repository, without worrying about any outside dependencies that she has no control over. Susie decides that her best bet is to install a static analysis tool. She decides to install FindBugs because she likes the quality of the results and the fact that bugs can be found as she types; at first, she is very happy with her decision and feels productive when using it.

The above scenario is an interpretation of an experience one of our participants recalled during their interview. Static analysis tools use well-defined programming rules to find defects early in the development process, when they are cheap to fix [6]. For example, there are static analysis tools that can alert developers to synchronization issues which can lead to unsafe thread interactions. Developers have been able to eliminate many defects that were previously overlooked at large companies [9] using the warnings produced by static analysis tools.

Despite the benefits of using static analysis tools to find bugs, consistent usage of these tools is not very frequent [6]. Remember Susie, who adopted a static analysis tool to improve the quality of her code? After using the tool for a while, dealing with the interface became a burden; finding the warnings was not easy and when she did, she had a hard time interpreting the feedback. Inspecting her code without using the tool involved more work, but she prefered to do it this way to avoid the time and confusion involved with using the tool. There have been studies to investigate ways of improving static analysis tools. However, none look at what the tools do or can do for a developer, what features developers use, what could be improved *and* why [10], [11]. Our research aims to understand why software developers are not using static analysis tools and how current tools could be improved to increase usage based on developer feedback. For our study, we intend to focus on static analysis tools used to finds bugs. This includes tools like FindBugs, Lint [12] , IntelliJ [4] (which includes built-in static analyzers), and PMD [13]. FindBugs will be referenced the most as it is the tool we chose to use during our interviews.

Johnson, Song, Murphy-Hill, Bowdidge. "Why Don't Software Developers Use Static Analysis Tools to Find Bugs?" ICSE 2013.

Tricorder: Static Analysis at Google



- Platform that runs many analyzers on every code change
- Results show up **in code review**, where the developer already is
- Every result has a **"Not useful"** button
- Analyzers with too many "Not useful" clicks get **fixed or turned off**
- Bar for shipping an analyzer: **under 10% effective false positives**

Sadowski, van Gogh, Jaspan, Söderberg, Winter. "Tricorder: Building a Program Analysis Ecosystem." ICSE 2015.

Definition: Effective False Positive

- A result is an **effective false positive** if the developer **took no positive action** after seeing it
- It doesn't matter whether the tool was *technically* right
- If developers don't act on it, it's noise
- The developer's response decides what counts as a false positive, not the tool author

Paraphrased from Sadowski et al., "Lessons from Building Static Analysis Tools at Google," CACM 61(4), 2018.

A page nobody acts on is an effective false positive

Lessons That Transfer to Alerting

Tricorder		Your alerting
"Not useful" button	→	"Was this page useful?" on every page
Measure fixes vs. dismissals	→	Track action-taken rate per alert rule
Disable noisy analyzers	→	Demote or delete noisy alerts
Analyzer owner fixes the message	→	Alert owner keeps the runbook current

Participation Activity

Page, ticket, or log? For each Tartan Tickets alert rule, decide where it should go

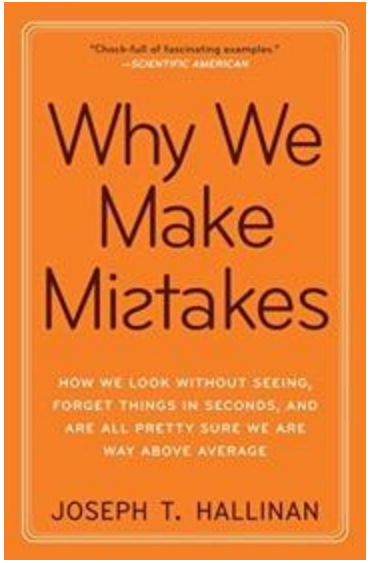
1. CPU > 80% on any ticket-api pod for 5 min
2. Checkout error rate > 5% for 5 min
3. TLS certificate expires in < 14 days
4. p99 purchase latency > 4s for 10 min
5. Any container restarted
6. Disk > 75% full on postgres volume
7. Zero successful purchases in 15 min during an on-sale
8. Canary error rate > 2× baseline while a flag is on

1. Think on your own (1 min)
2. Compare with your neighbor (3 min)
3. **Bonus:** which **page** would you also automate, so the human is only paged if the automation fails?

Activity Debrief

- **Page:** 2, 4, 7 (symptoms users feel)
- **Page, after automation:** 8, auto-abort the canary first, then page if that fails
- **Ticket:** 3, 6 (real, but they can wait until morning)
- **Log:** 1, 5 (causes, not symptoms)

Why We Make Mistakes



Generalization

- …in the words of psychologist Tom Stafford, we can't find our typos because we're engaging in a high-level task in writing. Our brains generalize simple, component parts to focus on complex tasks, so essentially we can't catch the small details because we're focused on a large task.

https://medium.com/swlh/why-we-miss-our-own-typos-96ab2f06afb7

Boredom

“Boredom can give rise to errors, adverse patient events, and decreased productivity—costly and unnecessary outcomes for consumers, employees, and organizations alike. As a function of boredom, individuals may feel over-worked or under-employed, and become distracted, stressed, or disillusioned. Staff who are bored also are less likely to engage with or focus on their work.”



Original Articles

Boredom in the Workplace: Reasons, Impact, and Solutions

Michelle Cleary , PhD, RN, Jan Sayers , PhD, RN, Violeta Lopez , PhD, RN & Catherine Hungerford , PhD, RN

Pages 83-89 | Received 24 Jun 2015, Accepted 13 Aug 2015, Published online: 10 Feb 2016

Download citation https://doi.org/10.3109/01612840.2015.1084554

Full Article | Figures & data | References | Citations | Metrics | Reprints & Permissions | Get access

Abstract

Boredom in the workplace is not uncommon, and has been discussed widely in the academic literature in relation to the associated costs to individuals and organizations. Boredom can give rise to errors, adverse patient events, and decreased productivity—costly and unnecessary outcomes for consumers, employees, and organizations alike. As a function of boredom, individuals may

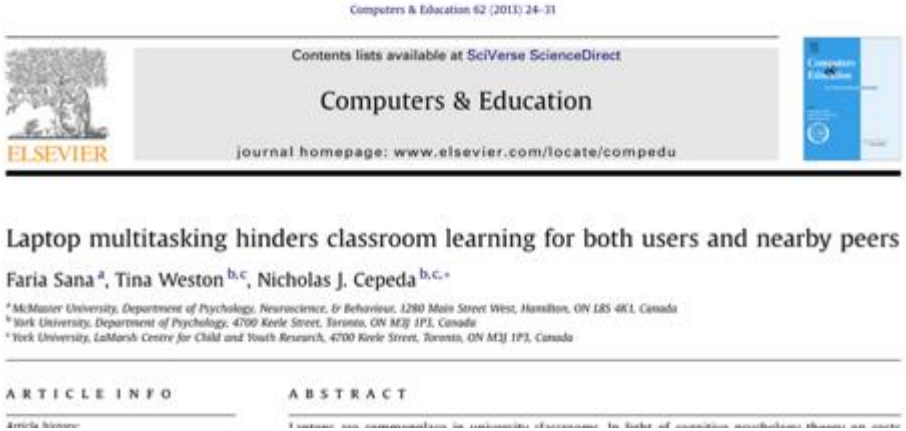
Related rese

People also read

Boredom at work spillover model work motivation boredom

Cognitive Load

- ...” students who switch back and forth between attending to a classroom lecture and checking e-mail, Facebook, and IMing with friends”



Computers & Education 62 (2013) 24–31

Contents lists available at SciVerse ScienceDirect

Computers & Education

journal homepage: www.elsevier.com/locate/compedu

Laptop multitasking hinders classroom learning for both users and nearby peers

Faria Sana [a], Tina Weston [b,c], Nicholas J. Cepeda [b,c,*]

[a] McMaster University, Department of Psychology, Neuroscience, & Behaviour, 1280 Main Street West, Hamilton, ON L8S 4K1, Canada
[b] York University, Department of Psychology, 4700 Keele Street, Toronto, ON M3J 1P3, Canada
[c] York University, LaMarsh Centre for Child and Youth Research, 4700 Keele Street, Toronto, ON M3J 1P3, Canada

ARTICLE INFO

ABSTRACT

Can we remove human error?

catch

Can we ~~remove~~ human error?

Can we catch human error before we ship our code?

Can we automate tasks to prevent problems?





Distinctio sexta Tractatus secundus.

Da notare circa a la figura ꝯ dicio posta ꝯ ch̄ sin sopdi li numeri ō la linea prima
quale comença da.1.e na fin a.10.dicendo cosi.1.2.3.4.5.6.7.8.9.10.sel nūero a
secondo spacio al numero del primo referirai haueral la prima specie de la pro
portiō multiplici cioe tupla.e se al primo referirai al numero del terco spacio ha
rai la seconda specie de la multiplici cioe tripla.E cosi seguendo in tutte le altre rige de los
to el simile troouerai.Da se al numero del secondo spacio el numero del terco spacio
rai cioe.3.a.2 haueral la prima specie de la proportione superparticulare cioe sequia
E se al terco el quarto cioe.4.a.3.sera quaterna.E se al quarto lo quito cioe.5.a.4.
quarta e cosi in altre sequi.Da se al numero di terco spacio compererai el numero del
to spacio cioe.5.a.3.haueral la prima specie de la proportione superpartiente cioe super
particulo tertias.E se al numero del quarto spacio el numero del septimo referirai
.7.a.4.haueral la seconda specie de la proportione superpartiēte cioe supertripartiēs
tas.E se al numero del quinto el nūo del nono cioe nouc a.5.farai la terça specie de la
partiēte cioe super quadripartiēs quintas.E se al numero del secondo el numero del
to cioe.5.a.2.farai la prima specie de la proportione multiplici superparticulare cioe
sesqualtera.E se a qual medesimo el septimo cioe.7.a.2.tripla sequialtera.Da se al
ro del terco si compara el numero de octauo cioe.8.a.3.sira la prima specie de la propo
ne multiplici superpartiēte cioe tupla superbipartiēs tertias .E cosi porrai p̄ te più
cedere se la tauola sira magiore.E se al quarto undecimo quando si fosse tacesti
specie detta tupla superquadripartiēs quartas cioe.11.quando più oltra volesi procede
E cosi commo habiamo detto de li numeri posti nela prima riga secondo li medesi
ghi comparādo ancora le linee inferiori quelle medesime specie te saranno che
la prima hai hauute.opero tu per te sequirai.7c.

Del termino elqual se usa in denominare molte specie de proportioni
qui non importa altro (a te pratico) sēnō a più cōmodamēte proferire
cie trouato.Et est vt supra de multiplicando integros numeros dicim
dam sylabica adiectio.Si cōmo decimo del via e del sia che sufano al
carc 7c.Da el sub ch̄ a cauare lespecie ō la menore inequalita si prepone a quelle
giore inequalita.E sī mera spolitio e cosi li super in più specie interposto.iō nō mir

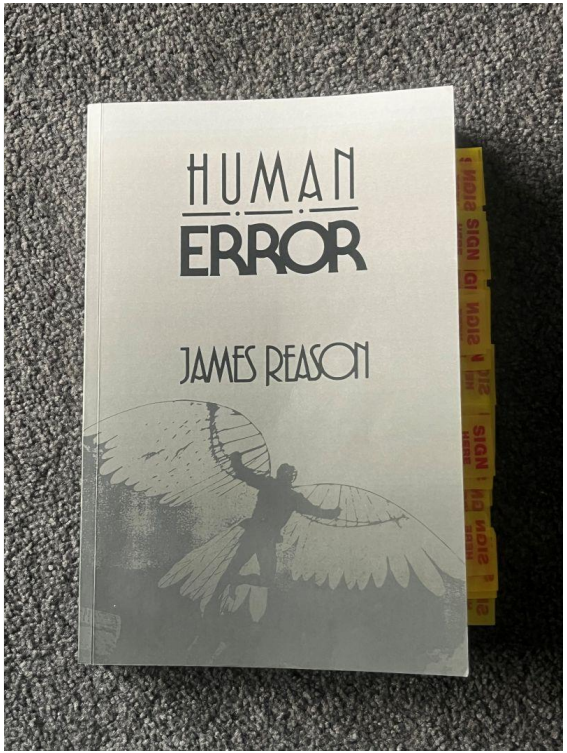
De proportionalitatibus tra.2°.sexte distin.arti.p°.

Auendo a bastanca de le proportioni parlato e quelle diuise fine a le

Approach:
Automate what we can
Review what we cannot

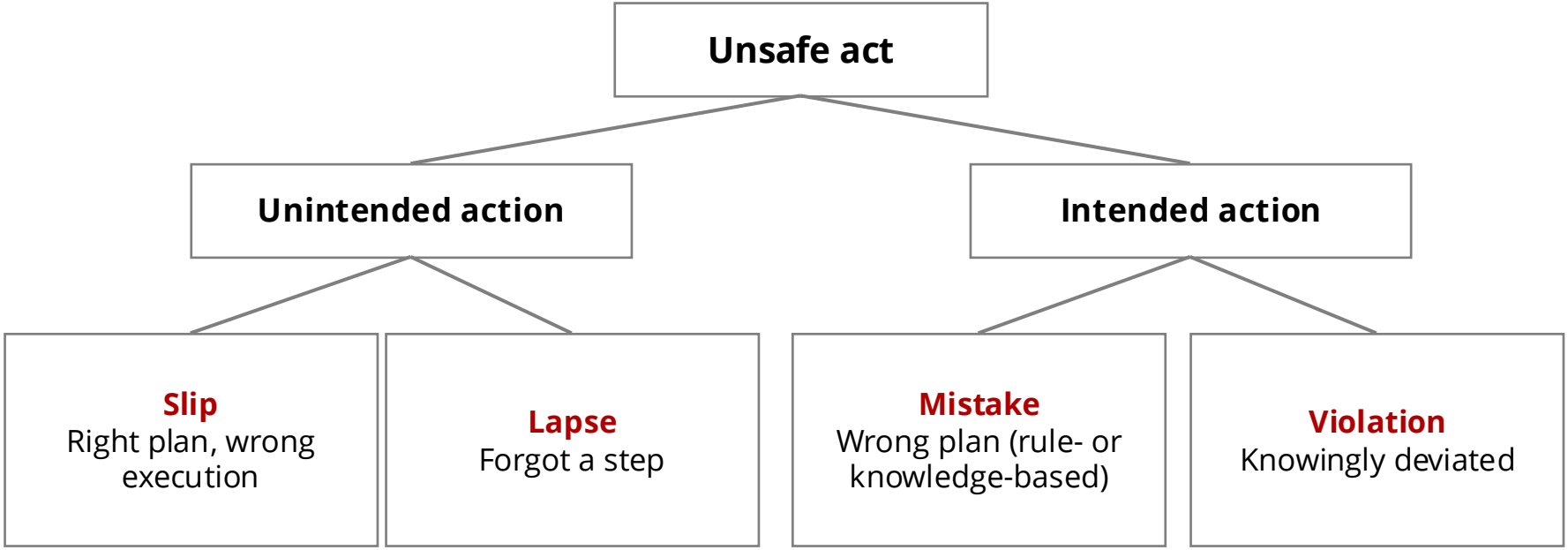
Human Error Is Normal, and Predictable

- Errors aren't random: they fall into **predictable types**
- Different types need **different defenses**
- "Be more careful" is not a defense
- "Blame the human" fixes nothing and teaches people to hide errors



James Reason. Human Error. Cambridge University Press, 1990.

Types of Error (Reason)



After James Reason, Human Error (1990).

Slip: Right Plan, Wrong Execution

- **AWS S3, February 28, 2017**
- An engineer debugging a billing system ran a command from an **established playbook**
- One input was mistyped; far more servers were removed than intended
- S3 in us-east-1 went down for hours, and much of the internet with it



amazon web services | Products | Solutions | Pricing | Software | Support | Customers | Partners | English | My Account | Create an

Summary of the Amazon S3 Service Disruption in the Northern Virginia (US-EAST-1) Region

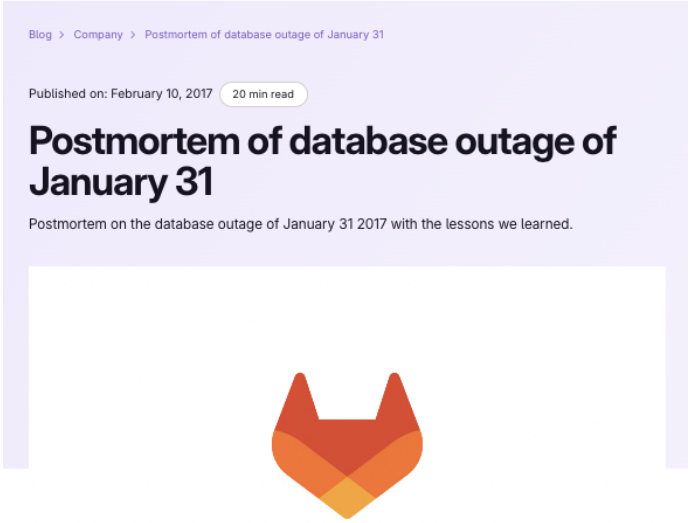
We'd like to give you some additional information about the service disruption that occurred in the Northern Virginia (US-EAST-1) Region on the morning of February 28th. The Amazon Simple Storage Service (S3) team was debugging an issue causing the S3 billing system to progress more slowly than expected. At 9:37AM PST, an authorized S3 team member using an established playbook executed a command which was intended to remove a small number of servers for one of the S3 subsystems that is used by the S3 billing process. Unfortunately, one of the inputs to the command was entered incorrectly and a larger set of servers was removed than intended. The servers that were inadvertently removed supported two other S3 subsystems. One of these subsystems, the index subsystem, manages the metadata and location information of all S3 objects in the region. This subsystem is necessary to serve all GET, LIST, PUT, and DELETE requests. The second subsystem, the placement subsystem, manages allocation of new storage and requires the index subsystem to be functioning properly to correctly operate. The placement subsystem is used during PUT requests to allocate storage for new objects. Removing a significant portion of the capacity caused each of these systems to require a full restart. While these subsystems were being restarted, S3 was unable to service requests. Other AWS services in the US-EAST-1 Region that rely on S3 for storage, including the S3 console, Amazon Elastic Compute Cloud (EC2) new instance launches, Amazon Elastic Block Store (EBS) volumes (when data was needed from a S3 snapshot), and AWS Lambda were also impacted while the S3 APIs were unavailable.

S3 subsystems are designed to support the removal or failure of significant capacity with little or no customer impact. We build our systems with the assumption that things will occasionally fail, and we rely on the ability to remove and replace capacity as one of our core operational processes. While this is an operation that we have relied on to maintain our systems since the launch of S3, we have not completely restarted the index subsystem or the placement subsystem in our larger regions for many years. S3 has experienced massive growth over the last several years and the process of restarting these services and running the necessary safety checks to validate the integrity of the metadata took longer than expected. The index subsystem was the first of the two affected subsystems that needed to be restarted. By 12:26PM PST, the index subsystem had activated enough capacity to begin servicing S3 GET, LIST, and DELETE requests. By 1:18PM PST, the index subsystem was fully recovered and GET, LIST, and DELETE APIs were functioning normally. The S3 PUT API also required the placement subsystem. The placement subsystem began recovery when the index subsystem was functional and finished recovery at 1:54PM PST. At this point, S3 was operating normally. Other AWS services that were impacted by this event began recovering. Some of these services had accumulated a backlog of work during the S3 disruption and required additional time to fully recover.

https://aws.amazon.com/message/41926/

Slip: Right Plan, Wrong Execution

- **GitLab.com, January 31, 2017**
- Late at night, mid-incident, an engineer ran a delete on the **wrong database host**
- Then the team found that the backup procedures weren't working
- About six hours of production data were lost



https://about.gitlab.com/blog/2017/02/10/postmortem-of-database-outage-of-january-31/

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

Lapse: Forgot a Step

- **Knight Capital, August 1, 2012** (recall the Sep 23 handout)
- New code was deployed to **7 of 8** servers
- The eighth server ran old, dormant code against live orders
- About **$460M** lost in 45 minutes
- Nothing in the process checked that every server was updated



SEC Administrative Proceeding, File No. 3-15570 (2013).

Mistakes: The Plan Itself Was Wrong

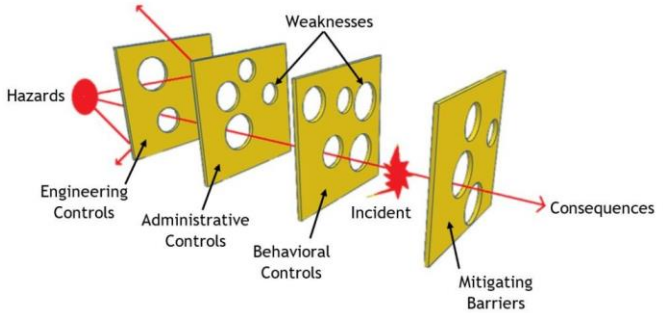
- **Rule-based:** applied a familiar fix to the wrong situation
  - *"Restarting fixed it last time"*
  - Diagnosis matched the pattern, but it was a different problem
- **Knowledge-based:** a novel failure where no rule exists
  - Reasoning from first principles, under pressure, at 3 a.m.
  - The hardest kind, and the one practice helps most

Violations: Knowingly Skipping a Step

- Usually well-intentioned:
  - *"The deploy checklist takes 20 minutes and prod is down"*
- A violation is a **signal**: the right way is too expensive
- Fix the process, not just the person
- **Make the right way the easy way**

The Swiss Cheese Model

- Every defense layer has holes
- Incidents happen when the holes **line up**
- Ask "which layers failed?", not "who failed?"
- GitLab: a slip **and** failed backups **and** no tested restore



James Reason. "Human error: models and management." BMJ 320(7237), 2000.

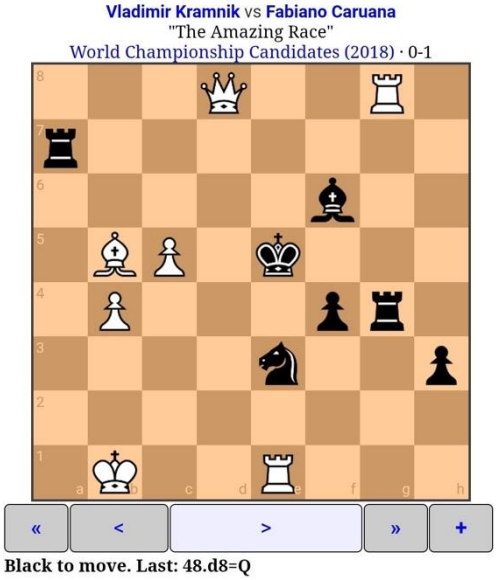
Match the Defense to the Error

Error	Defense
Slip	Guardrails: confirmations, rate limits, "type the hostname to continue"
Lapse	Checklists, automation, verification steps
Rule-based mistake	Runbooks that say how to **confirm** the diagnosis first
Knowledge-based mistake	Training, gamedays, fast escalation
Violation	Make the right way the easy way; blameless culture

Cognitive Load Under Stress

Your Working Memory Is Tiny

- Working memory holds about **4 chunks** at once
- Stress narrows attention: **tunnel vision**
- We fixate on the **first hypothesis** and ignore contradicting evidence
- Waking up and switching context costs more
- Every fact you must *remember* during an incident takes a slot



Cowan, "The magical number 4 in short-term memory," Behavioral and Brain Sciences 24(1), 2001.

Who Is On Call at 3:12 a.m.?

- Was asleep 90 seconds ago
- Didn't write this service
- Maybe their second week on the team
- Slack is exploding; a manager is asking for an ETA
- **This is the person your runbook is written for**

Do the thinking in daylight

Decide Before the Emergency

- What counts as an incident, and **who is in charge**
- Which mitigations are **safe without asking** (roll back, flag off)
- Who to escalate to, and **when**
- How and how often to **communicate status**
- Google SRE: playbooks give **roughly 3× faster recovery (MTTR)** than "winging it"

Beyer et al., Site Reliability Engineering, Introduction https://sre.google/sre-book/introduction/

Checklists (and Airplanes)

October 30, 1935: Boeing Model 299

- Prototype of the B-17, in a US Army Air Corps competition
- Crashed right after takeoff
- Cause: the crew had not released a **control lock**
- Flown by the Army's chief of flight testing
- Press verdict: *"too much airplane for one man to fly"*



Atul Gawande. The Checklist Manifesto. Metropolitan Books, 2009.

The Fix: Not More Training. A Checklist

- Test pilots wrote a short card: takeoff, flight, landing, taxi
- The B-17 went on to fly **1.8 million miles** without an accident (per Gawande)
- Expert pilots didn't need more skill
- They needed **help with the dumb stuff**, so their skill could go to the hard stuff



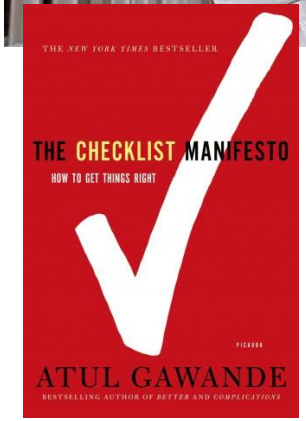


Atul Gawande. The Checklist Manifesto. Metropolitan Books, 2009.

Checklists in Surgery

- WHO Surgical Safety Checklist: 19 items, three pause points
- Tested in **8 hospitals** in 8 countries
- Deaths: **1.5% → 0.8%**
- Inpatient complications: **11.0% → 7.0%**
- Simple items: confirm patient identity, site, introduce the team by name





The Pronovost Checklist

Central venous catheters, or lines, are used for medications, blood, fluids or nutrition and can stay in for days or weeks. But bacteria can grow in the line and spread a type of infection to the bloodstream, which causes death in one out five patients who contract it. This five-step checklist for doctors and nurses to use before inserting a line can prevent infections and death.

1. **Wash hands** with soap and water or an alcohol cleanser
2. **Wear sterile clothing**—a mask, gloves, and hair covering—and cover patient with a sterile drape, except for a very small hole where the line goes in
3. **Clean patient's skin** with chlorhexidine (a type of soap) when the line is put in
4. **Avoid veins in arm and leg,** which are more likely to get infected than veins in chest
5. **Check the line for infection** each day and remove when no longer needed

Source: Dr. Peter Pronovost

Haynes et al. "A Surgical Safety Checklist to Reduce Morbidity and Mortality in a Global Population." NEJM 360:491-499, 2009.

Two Kinds of Checklists

Read-Do

- Read an item, then do it
- For **rare or unfamiliar** procedures
- e.g., database failover
- e.g., rotating a leaked secret

Do-Confirm

- Do the work from memory, then **pause** and confirm
- For **familiar, routine** work
- e.g., pre-deploy checks
- e.g., closing out an incident

What Makes a Good Checklist

- A clear **pause point**: when do you use it?
- Short: about **5–9 items**
- Only the **killer items**: steps that are dangerous to skip and easy to miss
- Exact, simple wording; fits on one page
- **Tested in real use** and revised

Daniel Boorman (Boeing), as described in Gawande, The Checklist Manifesto, 2009.

Checklists Support Judgment

- **US Airways 1549, January 15, 2009**
- Bird strike took out both engines at low altitude
- The dual-engine-failure checklist assumed **much higher altitude**
- Crew prioritized: started the APU early, flew the plane, landed on the Hudson
- All **155** people survived



NTSB Aircraft Accident Report NTSB/AAR-10/03, 2010.

Aviate, Navigate, Communicate

Pilots		On call
Aviate Keep the plane flying	→	**Mitigate** Stop the bleeding: roll back, flag off, shed load
Navigate Figure out where to go	→	**Diagnose** Find the cause, once users are safe
Communicate Talk to ATC	→	**Communicate** Status updates, from someone not typing commands

Checklists in DevOps

- **Pre-deploy** (do-confirm)
- **Incident start** (read-do)
- **Database failover** (read-do)
- **Launch readiness** for a new service
- **Incident close-out** (do-confirm)

```
# Incident start (read-do)
[ ] Declare the incident in #incidents
[ ] Name an incident commander
[ ] Name a comms lead (not the IC)
[ ] Check: deploy/flag change in last 1h?
    -> if yes, roll back FIRST
[ ] Post first status update (≤15 min)
[ ] Set the next update time
```

Runbooks

What Is a Runbook?

- Steps and context for **one specific situation**
- **Linked directly from the alert**
- Written in daylight, by someone who knows the system…
- …for someone who **doesn't**
- Also called: playbook, SOP, on-call guide

Anatomy of a Runbook

```
# Alert: CheckoutErrorRateHigh
## What it means
>5% of checkouts failing for 5 min.
Users cannot buy tickets.
## First 5 minutes
1. Deploy or flag change in last 30 min?
   -> roll back / flag off. Don't debug yet.
2. Check payment provider status page
## Diagnose
- Dashboard: errors by version  (link)
- Logs: checkout 5xx, last 15m   (link)
## Escalate
- Payments on-call if provider errors >50%
## Done when
- Error rate <1% for 15 min; post update
```

Meaning and user impact

Quick checks

Reversible mitigations first

Where to look (links!)

Escalation path

How you know you're done

Reversible Mitigations First

- **Roll back** to the last known good version
- **Flip the flag** off (recall Exposure Control, Oct 19)
- **Drain** the bad node; **shed** non-critical load
- Stop the bleeding *before* finding the cause
- A mitigation should be **safe to do half-asleep**

Runbooks Can Contain the Hazard

- Remember S3? The command came from an **established playbook**
- The runbook made the slip possible
- AWS's fix wasn't "be more careful":
  - Capacity is now removed **more slowly**
  - The tool **refuses** to go below a safe minimum
- Put guardrails **in the tool**, not just warnings in the doc

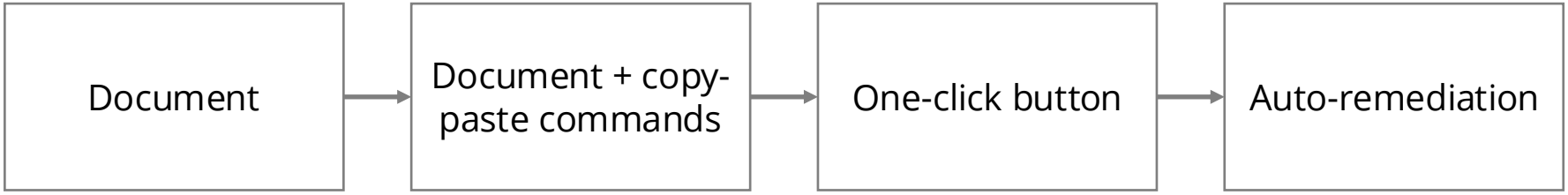
https://aws.amazon.com/message/41926/

A runbook nobody has run is a hypothesis

Runbook Rot

- Dashboards get renamed; links break
- Commands are for last year's cluster
- Backup procedures exist on paper but **don't actually restore** (recall GitLab)
- Fixes:
  - Every runbook has an **owner**
  - Update it **after every use**
  - **Practice** it before you need it → gamedays

From Document to Automation



← More flexible, needs a human

More consistent and faster →

Each step up the ladder removes a decision from 3 a.m.

Participation Activity

Write the first 5 minutes. Alert: `CheckoutErrorRateHigh` on Tartan Tickets

1. With your neighbor, write a **do-confirm** "first 5 minutes" runbook: **6 items max** (5 min)
2. **Swap** with the pair next to you
3. "Execute" their runbook against this scenario (3 min):

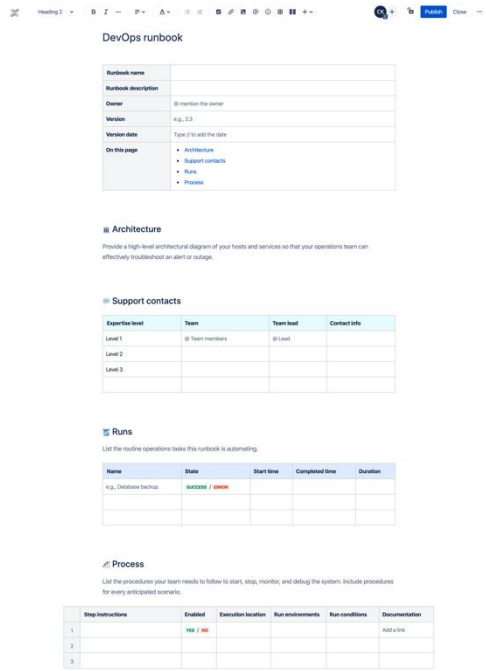
   It's 3:12 a.m. Checkout errors jumped to 12% at 2:58 a.m. The flag new-seat-picker went to 50% of users at 2:50 a.m. The payment provider's status page is green.

4. Did their runbook get you to the right mitigation? What was missing?

Runbooks for AI?

Agents Are Joining the On-Call Rotation

- AI agents increasingly **triage** alerts: gather logs, metrics, recent changes
- Some **summarize** and propose a mitigation
- Some are allowed to **act**
- For an agent, **the runbook is the instructions**



Writing a Runbook for an Agent?

- **Allowed actions:** read logs, query metrics, roll back to the last known good version
- **Forbidden actions:** delete data, change permissions, scale to zero
- **Stop conditions:** when to give up and page a human
- **Verification:** how to check the mitigation worked
- **Approval gates** = checklist pause points

Tired Humans vs. Literal Agents

On-call human

- Tired, slips and lapses
- Fixates on the first hypothesis
- Notices "this looks weird"
- Skips steps that look pointless

On-call agent

- Never tired, rarely lapses
- Confidently follows a **stale** runbook
- May not notice the situation doesn't match
- Does exactly what the words say

Reason's error types still apply, in **different proportions**

REMEMBER: Aviate, Navigate, Communicate

Pilots		On call
Aviate Keep the plane flying	→	**Mitigate** Stop the bleeding: roll back, flag off, shed load
Navigate Figure out where to go	→	**Diagnose** Find the cause, once users are safe
Communicate Talk to ATC	→	**Communicate** Status updates, from someone not typing commands

Would you let an agent run the S3 playbook?

Coming Up

Coming Up: Site Reliability Engineering

- Today's open question: **how bad is bad enough to page?**
- **SLOs** (service level objectives): how reliable users need us to be
- **Error budgets:** how much unreliability we can spend
- **Burn-rate alerts:** page when you're spending the budget too fast
- Reliability and Fault Injection (Mon, Nov 2)

Coming Up: Gamedays

- Break things **on purpose**, on a schedule, with everyone ready
- Find out whether the alerts fire, the runbooks work, and the people know what to do
- Applied Reliability and Chaos Engineering (Wed, Nov 4)
- Recovery and Learning (Wed, Nov 11)
- **Gameday Preparation** (Mon, Nov 23)

What You Need to Know



Page only when it's **urgent and actionable**; alert on symptoms



Effective false positives destroy trust: measure and prune



Errors have **types**; match the defense to the type



Do the thinking in daylight: checklists and runbooks

Questions?