

Confessions of a Systems Engineer

Learning from 20+ Years of Failures

David Argent

Amazon

Learning from Failure

“Develop success from failures. Discouragement and failure are two of the surest steppingstones to success.” -- Dale Carnegie

“It is fine to celebrate success, but it is more important to heed the lessons of failure.” -- Bill Gates

“Don’t be afraid to fail. Don’t waste energy trying to cover up failure. Learn from your failures and go on to the next challenge. It’s ok to fail. If you’re not failing, you’re not growing.” -- H. Stanley Judd

“Failure, the greatest teacher is.” -- Yoda



*When possible,
learn from
**OTHER PEOPLE'S
FAILURES** as well
as your own*



20+ Years of Wisdom from My Failures

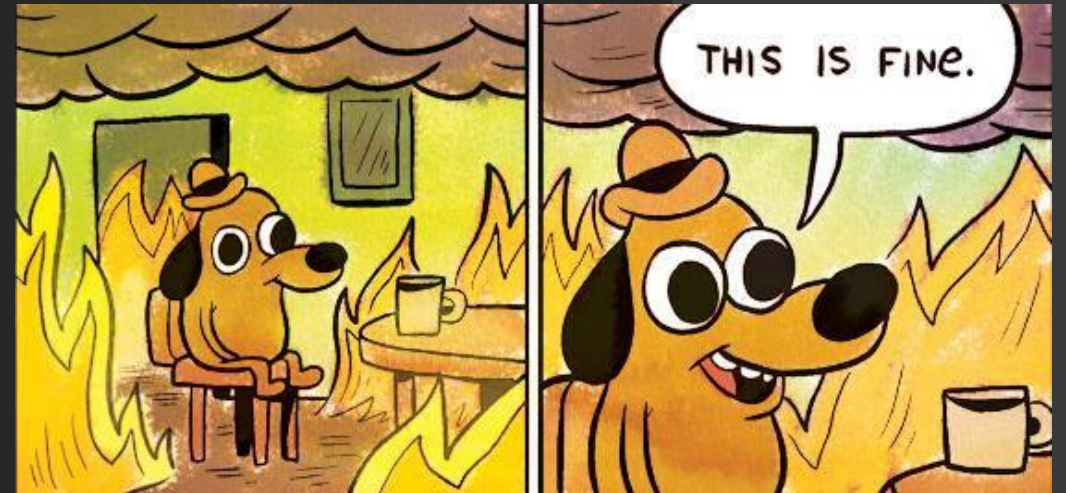
- There are no safe changes
- Minimize the blast radius on changes
- Monitor accurately and measure thoroughly
- Automate mitigations
- Anticipate failures and prepare with defined, degraded modes of service
- Use Functional Gates Pre-, Post-, and During Releases
- Design to meet SLA and mitigate incidents quickly
- Regularly exercise all processes and tools
- Enforce processes with technology
- Redirect traffic aggressively during incidents
- Tools used to maintain a service must be production quality
- Sanitize and verify inputs
- Understand all of the scenarios you support
- Transition service responsibilities carefully between groups

There Are No Safe Changes

- Any change is a potential incident waiting to happen
 - Processes change
 - Underlying code can be buggy
 - Dependencies can misbehave
 - Deployment automation can fail
 - Human error
 - The rest of Murphy's Law
- Code isn't the only thing that can take down your service
 - Code = Config = Data
 - Any of the above can kill your service
- Beware of cross-environment contamination in configs
 - Changes may apply to multiple environments
 - Not all "test" changes stay there, especially if default values are used

Key Lessons:

- Treat all changes with care and respect
- Code, config, or data - any change can take your service down

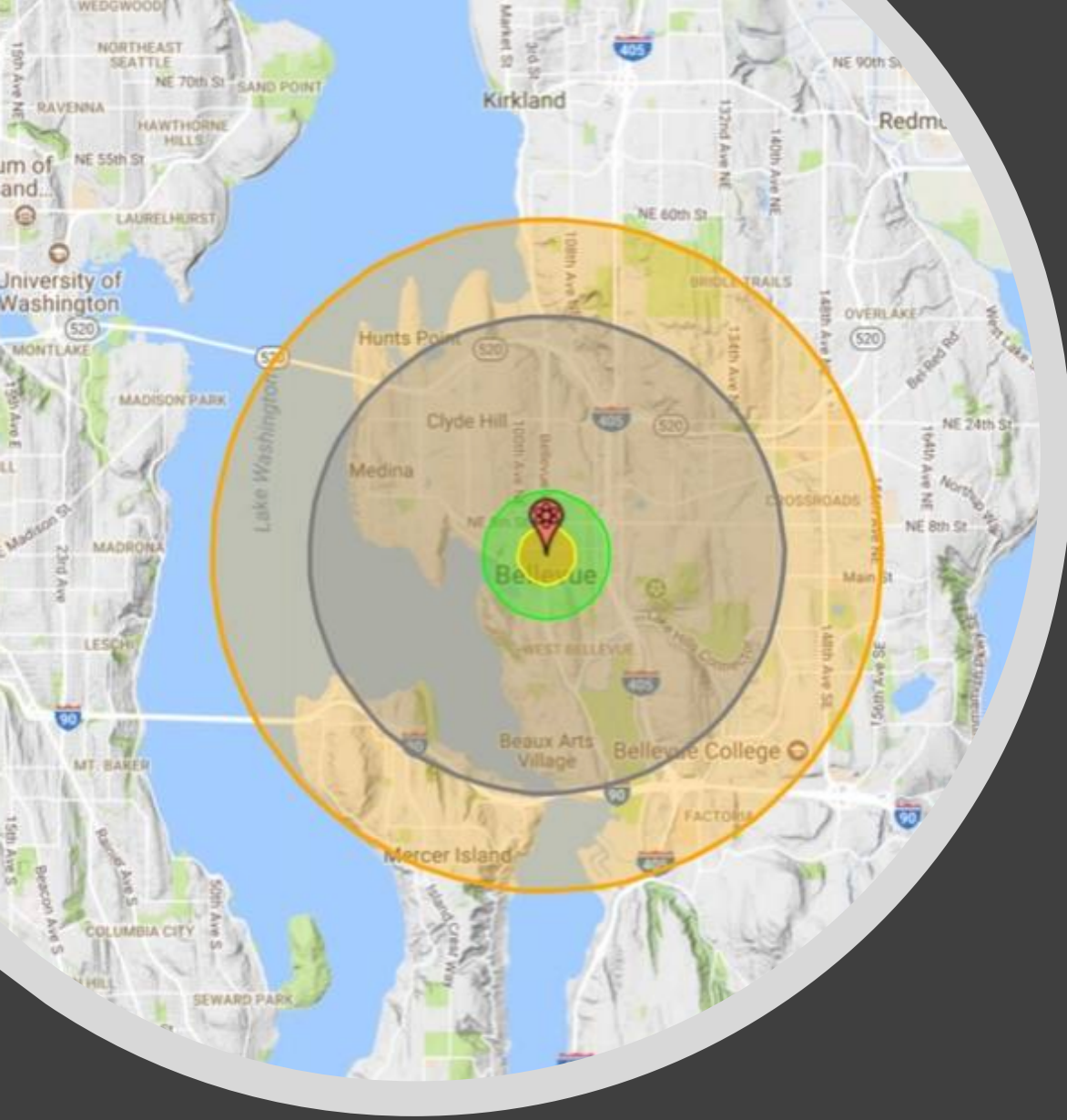


Minimize the Blast Radius on Changes

- Stage changes to progressively larger audiences
 - Stage to a small subset in a single DC first, then to a full DC, and then outward
 - Allows redirecting of traffic completely away from a dead DC at any point
 - Zero traffic on a DC enables faster rollback and more aggressive actions
- Positive validation is essential
 - Lack of negative feedback does not mean success
 - Sufficiently broken systems may provide NO feedback

Key Lessons:

- Risk reduction is key to successful change management
- Minimize the audience of changes and ramp up
- Balance speed and agility against risk as a conscious trade-off
- Lack of negative feedback cannot be your only signal it is safe to proceed





Monitor Accurately and Measure Thoroughly

- Understand your service during normal operation
 - Without understanding normal, you cannot understand broken
- Monitor accurately
 - Reduce false alarms to avoid unnecessary escalations
 - Direct escalations quickly to the person who can mitigate the issue
 - Make alerts informative enough to point to the underlying problem(s) quickly

Key Lessons:

- You can't know what's wrong, if you don't know what right looks like
- You can't automate a response if you can't programmatically see where there's a problem
- Direct escalations to the right person with sufficient information to quickly debug the issue if automated mitigation fails

Automate Mitigations

- Not all failures are “hard down” issues
 - Build systems to self-diagnose a wide range of failure modes
 - Attempt to repair failures and performance issues proactively
- Be aware of your dependencies and react to their failures
 - Automate your responses to dependency failures, as well as your own
- Speed matters
 - Automated responses are faster
 - Resolution times can be shorter than time to page an engineer

Key Lessons:

- No human being can react as quickly as a system that can diagnose itself accurately and take corrective actions
- Accurate monitoring and self-diagnosis of problems is key
- Reacting to partner or dependency issues can be as important as reacting to your own
- Measure. Mitigate. Automate. (MMA)



Degraded Service Modes, or An Imperfect Experience Usually Beats a Nonexistent One

- You will be unable to serve all your traffic, or a dependency will fail
 - Have tested SOPs in place for what to do
 - Don't rely on improvising a solution during a crisis
- Protect yourself from overload
 - Serving some customers is better than serving no customers
 - Have ways to shed excess load
 - Don't rely on partners or customers to self-limit load
 - Customers and partners are often the enemy of service availability
- Plan for failure
 - Work to provide your customer a useful experience, even during failures

Key Lessons:

- Design degraded service modes into your system
- Test your degraded modes and the SOPs surrounding their use

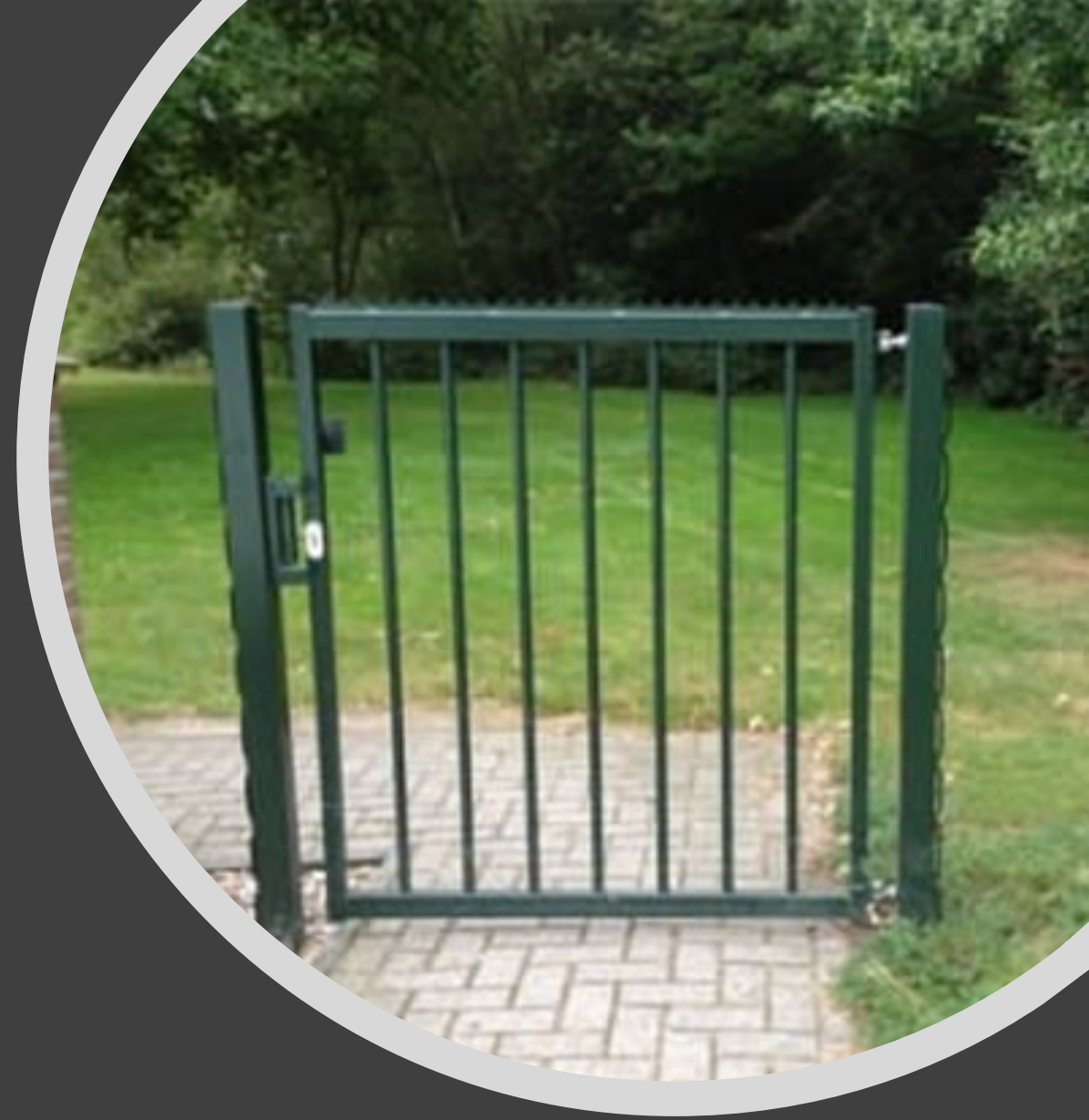


Use Functional Gates Pre-, Post-, and During Releases

- Good: Test before you deploy
- Better: Test after you deploy as well
- Best: Test during your deployment
 - Catch issues early
 - Proactively rollback while the blast radius is small
- Functional gates
 - Test and monitor actual service functionality
 - Accurately measure the customer experience
 - Synthetic tests are often inadequate
 - Specific monitoring against 'canary' and individual fault domains

Key Lessons:

- Test before, during, and after deployments
- Look at and monitor customer experience
- Have specific monitors specific to your fault domains



Not a very functional gate

Design to Meet SLAs and Mitigate Incidents Quickly

- Be aware of the limitations of your service and dependencies
 - Hard to design a single DC 99.999% service when the power to a single DC is only 99.99%
- Understand your SLA requirements
 - Designing for 5 9's is a lot harder and more expensive than designing for 3
- Design to be better than the sum of your parts and dependencies
 - Caching can reduce impact of dependency failures
 - Georedundancy can reduce impact of a DC failure
 - List not exhaustive
- Design with failure in mind
 - Mitigations for known failure modes must not break your SLA requirements
 - Often means having quick rollback options available

Availability %	Downtime per year	Downtime per month*	Downtime per week
90% ("one nine")	36.5 days	72 hours	16.8 hours
95%	18.25 days	36 hours	8.4 hours
97%	10.96 days	21.6 hours	5.04 hours
98%	7.30 days	14.4 hours	3.36 hours
99% ("two nines")	3.65 days	7.20 hours	1.68 hours
99.5%	1.83 days	3.60 hours	50.4 minutes
99.8%	17.52 hours	86.23 minutes	20.16 minutes
99.9% ("three nines")	8.76 hours	43.2 minutes	10.1 minutes
99.95%	4.38 hours	21.56 minutes	5.04 minutes
99.99% ("four nines")	52.56 minutes	4.32 minutes	1.01 minutes
99.999% ("five nines")	5.26 minutes	25.9 seconds	6.05 seconds
99.9999% ("six nines")	31.5 seconds	2.59 seconds	0.605 seconds

Key Lessons:

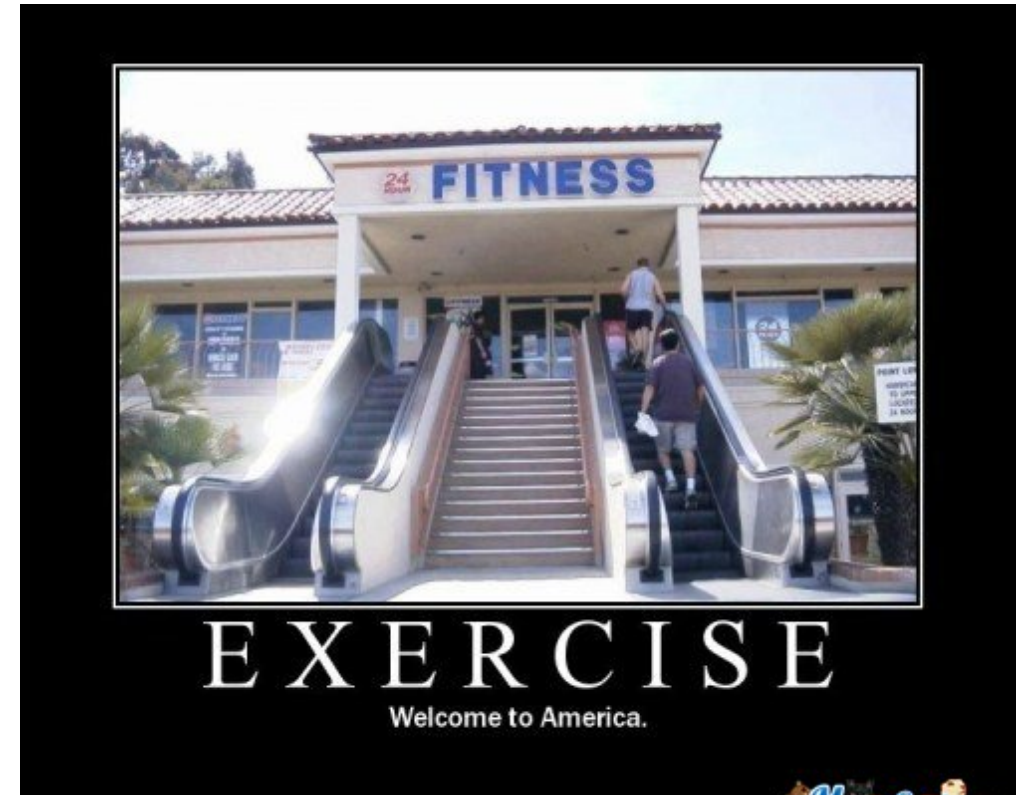
- Availability can be better than the sum of its parts, *if you design well*
- Understand your failure modes and how long they take to recover from – if a common failure takes you out of your SLA, you have a redesign ahead of you

Regularly Exercise All Processes and Tools

- An unverified backup is worse than no backup at all
- No disaster recovery exercise = no confidence
- Starting cold is not the same as starting warm
 - Can you warm up your cache before traffic overwhelms you?
 - Can you limit traffic to allow you to get back on your feet?

Key Lessons:

- An SOP you don't run regularly cannot be relied upon
- Recovery from hard down is often very different than recovering from a partial outage
- Design, tools, and processes need to work together to protect a service during a cold start



Enforce Processes with Technology

- Humans make mistakes
 - Good: Have processes in place to try to reduce human error
 - Great: Have technology in place to make it impossible
- Make the right way the only normal way
 - If a system only allows you to do a thing the right way, no human error
- Allow for 'higher risk' actions to deal with unanticipated incidents
 - Good guard rails are the enemy of flexibility during an incident
 - Balance ability to improvise and respond with risk with 2PA

Key Lessons:

- Always account for human error
- Make the easy way both the right way and the only typical way
- Understand when you need to take the guard rails off and why, with solid SOPs to help protect you from yourself



Gravity.

**It's not just a good idea.
It's the Law.**

Redirect or Drop Traffic Aggressively During Incidents

- An outage the customer might not notice beats one they definitely will
 - Slower, less complete experience >>> no experience at all
 - Service to some of your customers >>> service to none of your customers
- Redirect aggressively when a fault domain is underperforming
 - High latency or small availability drops can justify redirecting traffic
 - Low service quality impacts brand as well as bottom line
- Have objective criteria for what constitutes “unacceptable” service
 - Time to determine acceptable service quality isn’t during an incident

Key Lessons:

- Some service is usually better than no service
- Minimize the impact to the customer
- Decide ahead of time whether full service to some is better than degraded service to all and SOP accordingly
- Not all damage from an outage is from lost transactions



Production Quality Tools

- Any tool you depend on must itself be of Production quality
 - Incidents don't come on a convenient schedule
 - Nobody wants to mitigate only 99% of incidents
- Administrative tools can have a huge impact on a service
 - Rarely have the same rigor of test coverage
 - Infrequently exercised or tested

Key Lessons:

- Treat administrative tools as mission critical, because they are
- Exercise tools regularly, to ensure they're ready when needed
- Test your tools prior to every release – discovering the tools you rely on to operate your service don't work with the latest release is awkward when that release is in Production



Sanitize and Verify Inputs

- Trust no one – any data can be bad, despite best intentions
- All services communicating with yours will make a mistakes
- Malformed data isn't your friend, it's your conjoined twin you can never be rid of
- Everyone is a potential enemy of your service quality, including and especially yourself

Key Lessons:

- Design assuming you will have invalid data or requests
- Service configuration should receive extra validation, as a bad config parse can easily result in a complete service outage



**STAY
PARANOID
AND
TRUST
NO ONE**

Understand All of the Scenarios You Support

- It's not all just US Retail
- Understand who you are serving and their requirements
- Understand what traffic is important and why
 - This guides how you react during incidents
- Avoid cross region dependencies to isolate fault domains

Key Lessons:

- Design with data and service segregation in mind – avoid having one region's issues degrade service elsewhere
- Not all traffic is created equal
 - If you have the option to drop 'low priority' traffic during an adverse event, this may be the best option for your business
 - Recover 'high priority' traffic first during an adverse event where possible



Transition Service Responsibilities Carefully

- Constantly plan for transitions
 - This doubles as preparation for adding new team members
- Tribal knowledge is the enemy
 - Document and exercise your SOPs
 - Use your SOPs during every incident
 - Documentation remains up to date
 - Enables safer transitions of responsibility between groups
- Shadow existing staff during incidents for training opportunities

Key Lessons:

- It takes effort to run a service well – it also takes effort to teach someone else how to run a service well
- Document, document, document
- A good hand-off is a two-way transaction – merely saying a transition is “complete” does not mean it will be effective




You may imagine there was a copyrighted image here which showed how tribal knowledge is the enemy if a member of your team leaves the realm of the living suddenly and unexpectedly

Parting Thoughts

- No list like this is exhaustive, these are just some lessons from a long career spent operating online services
- Think about tomorrow, next month, and next year about what you'll need to operate your service at scale – once you fall behind, the technical debt to catch up can be difficult to surmount
- Prepare and design for success – one of the most dangerous times for a service is when it's embarrassingly successful
- Mistakes will happen, things will break – the best way to learn and improve is not to concentrate on assigning blame, but to work together on solutions to avoid similar situations moving forward

Questions? Anything Else?

Thanks for your time, I hope you can learn
from my mistakes



Feel free to contact me:
dargent@gmail.com

While the names may have been changed to protect the guilty, all of the advice and lessons here came from actual incidents that happened at a company with a corporate headquarters located near a large lake in Washington, where I worked for a QoS group.