# Learning from Learnings: Anatomy of Three Incidents

Randy Shoup
@randyshoup
linkedin.com/in/randyshoup

# Background





Google™

# App Engine Outage Oct 2012

- 4:00 am - Load begins increasing on traffic routers in one of the App Engine datacenters.
- 6:10 am - The load on traffic routers in the affected datacenter passes our paging threshold.
- 6:30 am - We begin a global restart of the traffic routers to address the load in the affected datacenter.
- 7:30 am - The global restart plus additional load unexpectedly reduces the count of healthy traffic routers below the minimum required for reliable operation. This causes overload in the remaining traffic routers, spreading to all App Engine datacenters. Applications begin consistently experiencing elevated error rates and latencies.
- 8:28 am - google-appengine-downtime-notify@googlegroups.com is updated with notification that we are aware of the incident and working to repair it.
- 11:10 am - We determine that App Engine's traffic routers are trapped in a cascading failure, and that we have no option other than to perform a full restart with gradual traffic ramp-up to return to service.
- 11:45 am - Traffic ramp-up completes, and App Engine returns to normal operation.

# App Engine Outage Oct 2012

In response to this incident, we have increased our traffic routing capacity and adjusted our configuration to reduce the possibility of another cascading failure.

Multiple projects have been in progress to allow us to further scale our traffic routers, reducing the likelihood of cascading failures in the future.

During this incident, no application data was lost and application behavior was restored without any manual intervention by developers. There is no need to make any code or configuration changes to your applications.

We will proactively issue credits to all paid applications for ten percent of their usage for the month of October to cover any SLA violations. This will appear on applications' November bills. There is no need to take any action to receive this credit.

http://googleappengine.blogspot.com/2012/10/about-todays-app-engine-outage.html

# App Engine Reliability Fixit

- Step 1: Identify the Problem
  - All team leads and senior engineers met in a room with a whiteboard
  - Enumerated all known and suspected reliability issues
  - Too much technical debt had accumulated
  - Reliability issues had not been prioritized
  - Identify 8-10 themes

# App Engine Reliability Fixit

- Step 2: Understand the Problem
  - Each theme assigned to a senior engineer to investigate
  - Timeboxed for 1 week
  - After 1 week, all leads came back with
    - Detailed list of issues
    - Recommended steps to address them
    - Estimated order-of-magnitude of effort (1 day, 1 week, 1 month, etc.)

# App Engine Reliability Fixit

- Step 3: Consensus and Prioritization
  - Leads discussed themes and prioritized work
  - Assigned engineers to tasks

# App Engine Reliability Fixit

- Step 4: Implementation and Follow-up
  - Engineers worked on assigned tasks
  - Simple spreadsheet of task status, which engineers updated weekly
  - Minimal effort from management (~1 hour / week) to summarize progress at weekly team meeting

# App Engine Reliability Fixit

- ➔ Results
  - 10x reduction in reliability issues
  - Improved team cohesion and camaraderie
  - Broader participation and ownership of the future health of the platform
  - Still remembered several years later

STITCH FIX™

# Stitch Fix Outages Oct / Nov 2016

- (11/08/2016) Spectre unavailable for ~3 minutes **[Shared Database]**
- (11/05/2016) Spectre unavailable for ~5 minutes **[Shared Database]**
- (10/25/2016) All systems unavailable for ~5 minutes **[Shared Database]**
- (10/24/2016) All systems unavailable for ~5 minutes **[Shared Database]**
- (10/21/2016) All systems unavailable for ~3 ½ hours [DDOS attack]
- (10/18/2016) All systems unavailable for ~3 minutes **[Shared Database]**
- (10/17/2016) All systems unavailable for ~20 minutes **[Shared Database]**
- (10/13/2016) Minx escalation broken for ~2 hours [Zendesk outage]
- (10/11/2016) Label printing unavailable for ~10 minutes [FedEx outage]
- (10/10/2016) Label printing unavailable for ~15 minutes [FedEx outage]
- (10/10/2016) All systems unavailable for ~10 minutes **[Shared Database]**

# Database Stability Problems

- 1. Applications contended on common tables
- 2. Scalability limited by database connections
- 3. One application could take down entire company

# Stitch Fix
# Stability Retrospective

- Step 1: Identify the Problem
- Step 2: Understand the Problem
- Step 3: Consensus and Prioritization
- Step 4: Implementation and Follow-Up
- ➔ Results

# Stability Solutions

- 1. Focus on expensive queries
  - Log
  - Eliminate
  - Rewrite
  - Reduce
- 2. Manage database connections via connection concentrator
- 3. Stability and Scalability Program
  - Ongoing 25% investment in services migration


we

# WeWork Login Issues

- *Problem: Some members unable to log in*
- Inconsistent representations across different services in the system
- Over time, simple system interactions grew increasingly complex and convoluted
- Not enough graceful degradation or automated repair

# WeWork
# Login Retrospective

- Step 1: Identify the Problem
- Step 2: Understand the Problem
- Step 3: Consensus and Prioritization
- Step 4: Implementation and Follow-Up

# Common Elements

- Unintentional, long-term accumulation of small, individually reasonable decisions
- "Compelling event" catalyzes long-term change
- Blameless culture makes learning and improvement possible
- Structured post-incident approach

If you don't end up regretting your early technology decisions, you probably over-engineered.

# Lessons

- Leadership and Collaboration
- Quality and Discipline
- Driving Change

# Lessons

- Leadership and Collaboration
- Quality and Discipline
- Driving Change

# Blameless Post-Mortems

- Open and Honest Discussion
  - Detect
  - Diagnose
  - Mitigate
  - Remediate
  - Prevent

➔ Engineers compete to take personal responsibility (!)

“Finally we can prioritize fixing that broken system!”

# Cross-Functional Collaboration

- Best decisions made through partnership
- Shared context
  - Tradeoffs and implications
  - Given common context, well-meaning people generally agree
- "Disagree and Commit"

"Never let a good crisis go to waste."

# Lessons

- Leadership and Collaboration
- Quality and Discipline
- Driving Change

# Quality and reliability are business concerns

"We don't have time to do it right!"

"Do you have time to do it **twice**?"

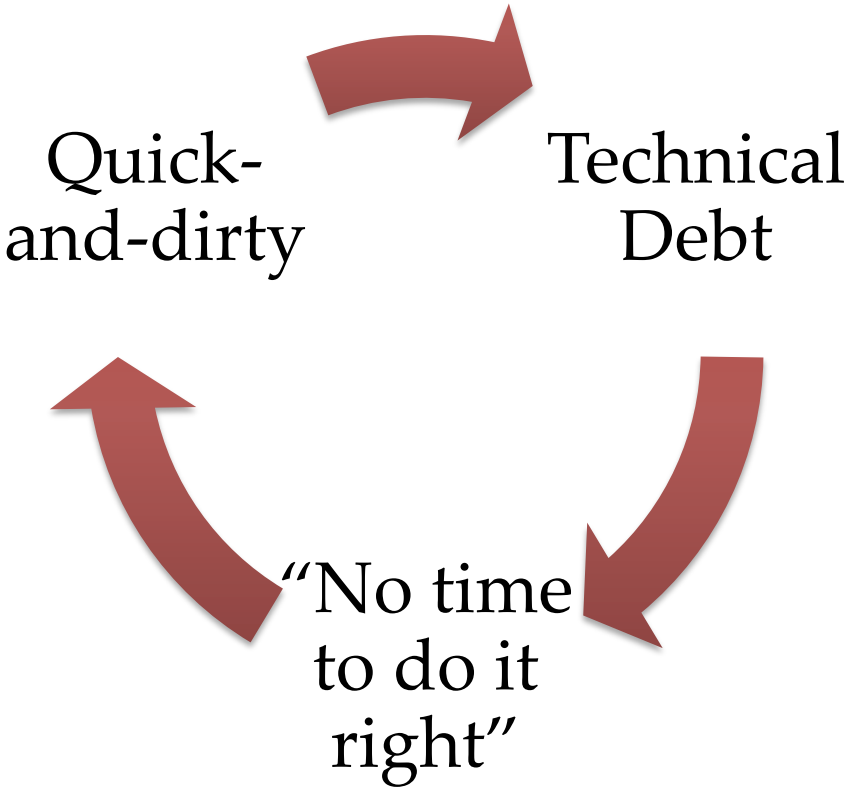
The more constrained you are on time or resources, the more important it is to build it right the first time.

"Do not try to do everything. Do one thing well."

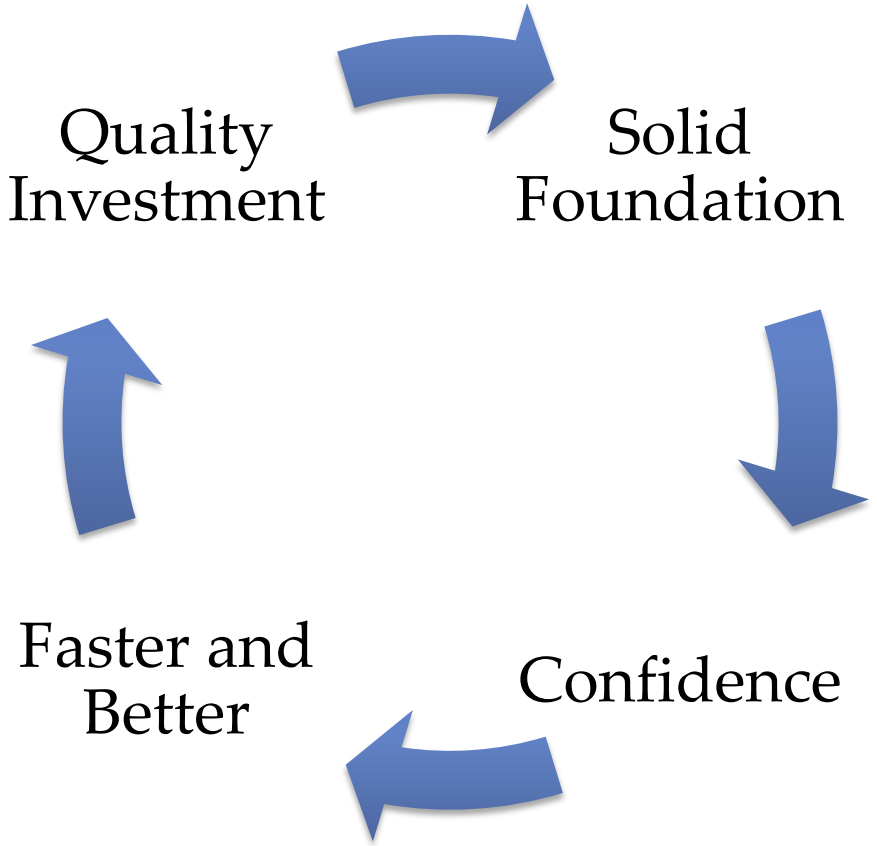
# Definition of Done

- Feature complete
- Automated Tests
- Released to Production
- Feature gate turned on
- Monitored

# Vicious Cycle of Technical Debt



# Virtuous Cycle of Investment



# Lessons

- Leadership and Collaboration
- Quality and Discipline
- Driving Change

# Top-down and Bottom-up

# Funding the Improvement Program

- Approach 1: Standard project
  - Prioritize and fund like any other project
  - Works best when project is within one team

- Approach 2: Explicit global investment
  - Agree on an up-front investment
    (e.g., 25%, 30% of engineering efforts)
  - Works best when program is across many teams

# An unexpected ally ...

# Finance

# Lessons

- Leadership and Collaboration
- Quality and Discipline
- Driving Change

"If you can't change your organization,
change your organization."

-- Martin Fowler

# We are Hiring!


700 software engineers globally, in

- New York
- Tel Aviv
- San Francisco
- Seattle
- Shanghai
- Singapore