“When two-pizza teams plan a banquet”
Lightweight architectural governance

Jonny LeRoy
@jahnnie

ThoughtWorks

oreillysacon.com
#OREillySACon
LIGHTWEIGHT ARCHITECTURE

GOVERNANCE
THE "G" WORD

GOVERNANCE
GOVERNANCE

κυβερνάω

1. Lat. gubernare, to steer, Od., etc.: absol. to act as pilot or helmsman, Ar.
2. metaph. to guide, govern, Pind., Soph.

κυβερνήτης κυβερνάω

1. a steersman, helmsman, pilot, Lat. gubernator, Hom., etc.: ionic acc. κυβερνήτης Hdt.
2. metaph. a guide, governor, Eur., Plat.
**GOVERNANCE**

**κυβερνάω**

1. Lat. guberna, to govern, pilot or helmsman, Ar.
2. metaph. to rule

**κυβερνήτης**

1. a steersman, helmsman, *gubernator*, Hom., etc.: ionic acc. to *gubernátēs*, Hdt.
2. metaph. a guide, governor, Eur., Plat.
WHY GOVERNANCE?
WHY GOVERNANCE?

Steer the ship

Avoid the rocks
WHY GOVERNANCE?

Alignment with strategy

Risk management
μServices

Autonomous “2-pizza” teams

Drone 'sheepdogs' steer flocks away from airports to prevent bird strikes

CBC Radio • August 24, 2018

Bird strikes number in the thousands per year and have resulted in incidents such as the one that inspired the movie "Miracle on the Hudson". (Getty images)
“When 2-pizza teams plan a banquet”

co-ordinated action in a complex and decentralized world
TWO FAILURE MODES

“Goldilocks zone”

Rigidity

Chaos
THE NEED FOR SPEED
THE NEED FOR SPEED

The department of “no”

No department
If you want to build a ship, don't drum up people to collect wood and don't assign them tasks and work, but rather teach them to long for the endless immensity of the sea.

Antoine de Saint-Exupéry
The Twelve Factors

I. Codebase
One codebase tracked in revision control, many deploys

II. Dependencies
Explicitly declare and isolate dependencies

III. Config
Store config in the environment

IV. Backing services
Treat backing services as attached resources

V. Build, release, run
Strictly separate build and run stages

VI. Processes
Execute the app as one or more stateless processes

VII. Port binding
Export services via port binding

VIII. Concurrency
Scale out via the process model

IX. Disposability
Maximize robustness with fast startup and graceful shutdown

X. Dev/prod parity
Keep development, staging, and production as similar as possible

XI. Logs
Treat logs as event streams

XII. Admin processes
Run admin/management tasks as one-off processes

1. Ship it and iterate.
2. Make changes small, make them often.
3. Technical debt is a useful tool.
4. Solve problems at the root.
5. Do not accept deviant system behaviour.
6. Write code to be read.
7. Write code to be debugged.
8. If you can’t show it’s a bottleneck, don’t optimise it.
9. Unblock others whenever you can.
10. Leave the codebase better than you found it.

https://monzo.com/blog/2018/06/29/engineering-principles/

https://12factor.net/
The Twelve Factors

I. Codebase
One codebase tracked in revision control, many deploys

II. Deployments
Explicit, automatic, and frequent deploys

III. Configuration
Store configuration carefully

IV. Build
Treat built software as the true architecture

V. Build
Strictly separate concerns

VI. Process
Execute processes promptly

VII. Export
Export your software

VIII. Scale
Scale out

IX. Deployments
Maximise availability

X. Design
Keep design simple

XI. Logs
Treat logs as event streams

XII. Admin processes
Run admin/management tasks as one-off processes

1. Start with user needs
2. Do less
3. Design with data
4. Do the hard work to make it simple
5. Iterate. Then iterate again
6. This is for everyone
7. Understand context
8. Build digital services, not websites
9. Be consistent, not uniform
10. Make things open: it makes things better

Ship it and iterate.
Make changes small, make them often.
Technical debt is a useful tool.
Solve problems at the root.
Do not accept deviant system behaviour.
Write code to be read.
Write code to be debugged.
If you can’t show it’s a bottleneck, don’t optimise it.
Unblock others whenever you can.
Leave the codebase better than you found it.

https://monzo.com/blog/2018/06/29/engineering-principles/
https://12factor.net/
**STRATEGIC GOALS**
Goals of the business side

- **REDUCE TIME TO MARKET**
  Establish fast feedback loops to learn, validate and improve. Remove friction, hand-offs and undifferentiated work.

- **SUPPORT DATA-DRIVEN DECISIONS**
  Provide relevant metrics and data for user and market insights. Validate hypothesis for problems worth solving.

- **MOBILE FIRST**
  Start small and use device capabilities.

- **BEST TALENT**
  Autonomy, Purpose and Mastery: We know why we do things, we decide how to approach them and deliberately practice our skills.

- **COST EFFICIENCY**
  Run your segment in the right balance of cost and value.

- **ONE SCOUT IT**
  Foster collaboration. Harmonize and standardize tools. Pull common capabilities into decoupled platform services.

**ARCHITECTURAL PRINCIPLES**
High-Level Principles

- **ORGANIZED AROUND BUSINESS CAPABILITIES**
  Build teams around products not projects. Follow the domain and respect bounded contexts. Make boundaries explicit. Inverse Conway Maneuver.

- **ELIMINATE ACCIDENTAL COMPLEXITY**
  Strive to keep it simple. Don’t over-engineer. Focus on necessary domain complexity.

- **LOOSELY COUPLED**
  By default avoid sharing and tight coupling. No integration database. Don’t create the next monolith.

- **MACRO AND MICRO ARCHITECTURE**
  Clear separation. Autonomous micro services within the rules and constraints of the macro architecture.

- **SECURITY, COMPLIANCE AND DATA PRIVACY**
  Build with least privilege and data privacy in mind. Know your threat model. Limit blast radius.

- **AWS FIRST**
  Favor AWS platform service over managed service, over self-hosted OSS, over self built solutions.

**DESIGN AND DELIVERY PRINCIPLES**
Tactical measures

- **YOU BUILT IT, YOU RUN IT**
  The team is responsible for shaping, building, running and maintaining its products. Fast feedback from live and customers helps us to continuously improve.

- **CROSS-FUNCTIONAL TEAMS**
  Engineers from all backgrounds work together in collaborative teams as engineers and share responsibilities. No silos.

- **AUTONOMOUS TEAMS**
  Make fast local decisions. Be responsible. Know your boundaries. Share findings.

- **BE BOLD**
  Go into production early. Value monitoring over tests. Fail fast, recover and learn. Optimize for MTTR not MTBF.

- **DATA-DRIVEN / METRIC-DRIVEN**
  Collect business and operational metrics. Analyze, alert and act on them.

- **INFRASTRUCTURE AS CODE**
  Automate everything: Reproducible, traceable, auditable and tested. Immutable servers.
ORGANIZATIONAL and SOFTWARE DELIVERY PERFORMANCE

COMMERCIAL
- Profitability
- Productivity
- Market share
- Number of customers

NON-COMMERCIAL
- Quantity of products/services
- Operating efficiency
- Customer satisfaction
- Quality of products/services
- Achieving organizational goals

THROUGHPUT
- Lead time
- Deployment frequency

STABILITY
- Mean time to restore (MTTR)
- Change fail percentage
SPEED + STABILITY no longer in conflict... but what about RISK?

THROUGHPUT
- Lead time
- Deployment frequency

STABILITY
- Mean time to restore (MTTR)
- Change fail percentage
AUTOMATE COMPLIANCE
SECURITY AUTOMATION

Archery
Zap
Snyk
Git-secrets
Quay.io
Grafeas
OpenSCAP
Brakeman
Find Security Bugs
...

...
OTHER DIMENSIONS

Quality
Performance
Resilience
Regulatory
Architectural characteristics
Cost
...
...
“BUT IT’S HARD
WE’RE NOT EXPERTS
WE’VE NEVER DONE THIS
(we don’t trust our people)
THE SECURITY SANDWICH
# The Sandwich Alignment Chart

<table>
<thead>
<tr>
<th>Structure</th>
<th>Ingredient Purist (Must have classic sandwich toppings: meat, cheese, lettuce, condiments, etc.)</th>
<th>Ingredient Neutral (Can contain a broader scope of savory ingredients)</th>
<th>Ingredient Rebel (Can contain literally any food products sandwiched together)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Purist</strong></td>
<td><strong>Hardline Traditionalists</strong></td>
<td><strong>Structural Purist, Ingredient Neutral</strong></td>
<td><strong>Structural Purist, Ingredient Rebel</strong></td>
</tr>
<tr>
<td>(A sandwich must have a classic sandwich shape: two pieces of bread/baked product, with toppings in between)</td>
<td>“A BLT is a sandwich.”</td>
<td>“A chip butty is a sandwich.”</td>
<td>“Ice cream between waffles is a sandwich.”</td>
</tr>
<tr>
<td><strong>Neutral</strong></td>
<td><strong>Structural Neutral, Ingredient Purist</strong></td>
<td><strong>True Neutral</strong></td>
<td><strong>Structural Neutral, Ingredient Rebel</strong></td>
</tr>
<tr>
<td>(The container must be on either side of the toppings, but not necessarily two separate pieces)</td>
<td>“A sub is a sandwich.”</td>
<td>“A hot dog is a sandwich.”</td>
<td>“An ice cream taco is a sandwich.”</td>
</tr>
<tr>
<td><strong>Rebel</strong></td>
<td><strong>Structural Rebel, Ingredient Purist</strong></td>
<td><strong>Structural Rebel, Ingredient Neutral</strong></td>
<td><strong>Radical Sandwich Anarchy</strong></td>
</tr>
<tr>
<td>(Can contain any food enveloped in any way by a containing food)</td>
<td>“A chicken wrap is a sandwich.”</td>
<td>“A burrito is a sandwich.”</td>
<td>“A Pop-Tart is a sandwich.”</td>
</tr>
</tbody>
</table>

---

**The Security Sandwich**
fixing THE SECURITY SANDWICH

- Automation
- Capability building
- Expert help
ENROLL GATEKEEPERS AS COLLABORATORS
FOR YOUR REFERENCE (ARCHITECTURE) (IMPLEMENTATION)
How do you extend the shelf-life of your reference architecture?
Your architectural mandates are really just hypotheses to test ...
from **FORK AND FORGET** to **FEEDBACK LOOPS**

harvest > predict
WE’RE ESCAPING THE MONOLITH
But where are we going after that?
PAVING the ROAD to the PIT OF SUCCESS
ONBOARDING, DEVELOPER EXPERIENCE and PRODUCT THINKING
SLAs for ARBs ...?
and architecture review as education opportunity
WE’RE NOT A SHIP
WE’RE A FLOTILLA
and we don’t have enough experts for each team
RETHINK your COMMUNICATION PATTERNS
1. Record architecture decisions

Date: 2016-02-12

Status

Accepted

Context

We need to record the architectural decisions made on this project.

Decision

We will use Architecture Decision Records, as described by Michael Nygard in this article: http://thinkrelevance.com/blog/2011/11/15/documenting-architecture-decisions

Consequences

See Michael Nygard’s article, linked above.
RETHINK your COMMUNICATION PATTERNS

North / South

East / West
- Bottom-up sharing of successes and failures

- Opportunity for governance / review

- Alignment around shared vision

- Balance standardization and innovation

- Inclusive and fun learning opportunity
STICKIES AT THE READY

CUSTOMIZE THE RINGS + QUADRANTS
End of life / retire
Packages, initiatives

RADAR OF RADARS
Radars per team, group, portfolio, organization

PROCESS / GUIDELINES FOR MOVING RINGS
Success criteria for assess, trial or adopt

WIP LIMITS
How much experimentation at one time?
What is your risk tolerance?
Architecture with 800 of My Closest Friends: The Evolution of Comcast’s Architecture Guild

Key Takeaways

- Modern software architecture in medium-to-large companies is increasingly a distributed affair. Agile methodologies, DevOps, and microservices have all enabled great independence for teams to make their own technical decisions.

- Many companies still rely on a tree-structured organizational structure for internal communications, often creating silos where it is difficult to discover what choices other teams are making.

- Comcast have cultivated an Architecture Guild, with the goal of “threading the needle” between obtaining advantageous critical mass around certain common technologies without undermining individual teams’ agency.

- The Architecture Guild is a grassroots framework that has been used to cut across organizational boundaries to identify solid, workable, default recommendations for technologies and practices explicitly modeled on existing successful decentralized groups like the IETF.

https://www.infoq.com/articles/architecture-guild-800-friends
THE MAJORhiftS REQUIRED

🌟 From MANDATE to VISION and PRINCIPLES

🌟 AUTOMATE COMPLIANCE

🌟 Enroll GATEKEEPERS as COLLABORATORS

🌟 RETHINK your COMMUNICATION PATTERNS

🌟 PRODUCT THINKING / SERVICE MENTALITY

🌟 COMFORT with EVOLUTION
how to get started

GO TO WHERE THE ACTION IS

START THE CONVERSATION
SERVANT LEADERSHIP, ARCHITECTURE GOVERNANCE and your TALENT STRATEGY
“When two-pizza teams plan a banquet”
Lightweight architectural governance

Jonny LeRoy