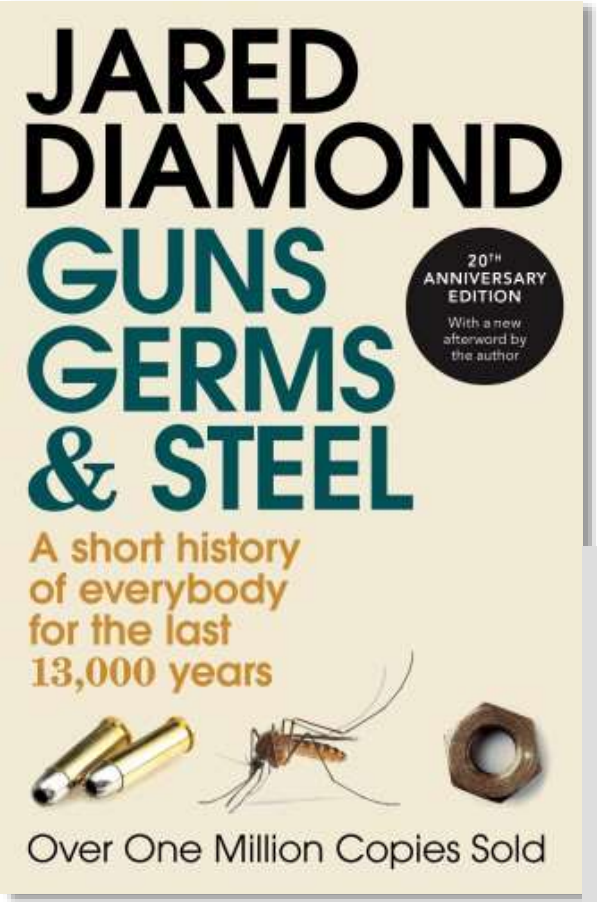
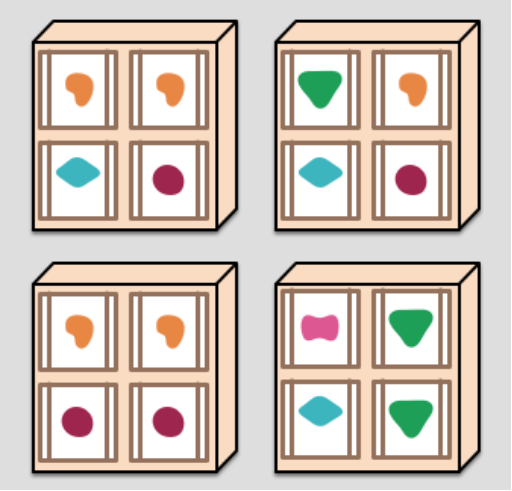
Guns, Germs, and Steel (and Microservices)

Daniel Bryant

@danielbryantuk





SpectoLabs

https://twitter.com/colettecello/status/749052871719591937

tl;dr

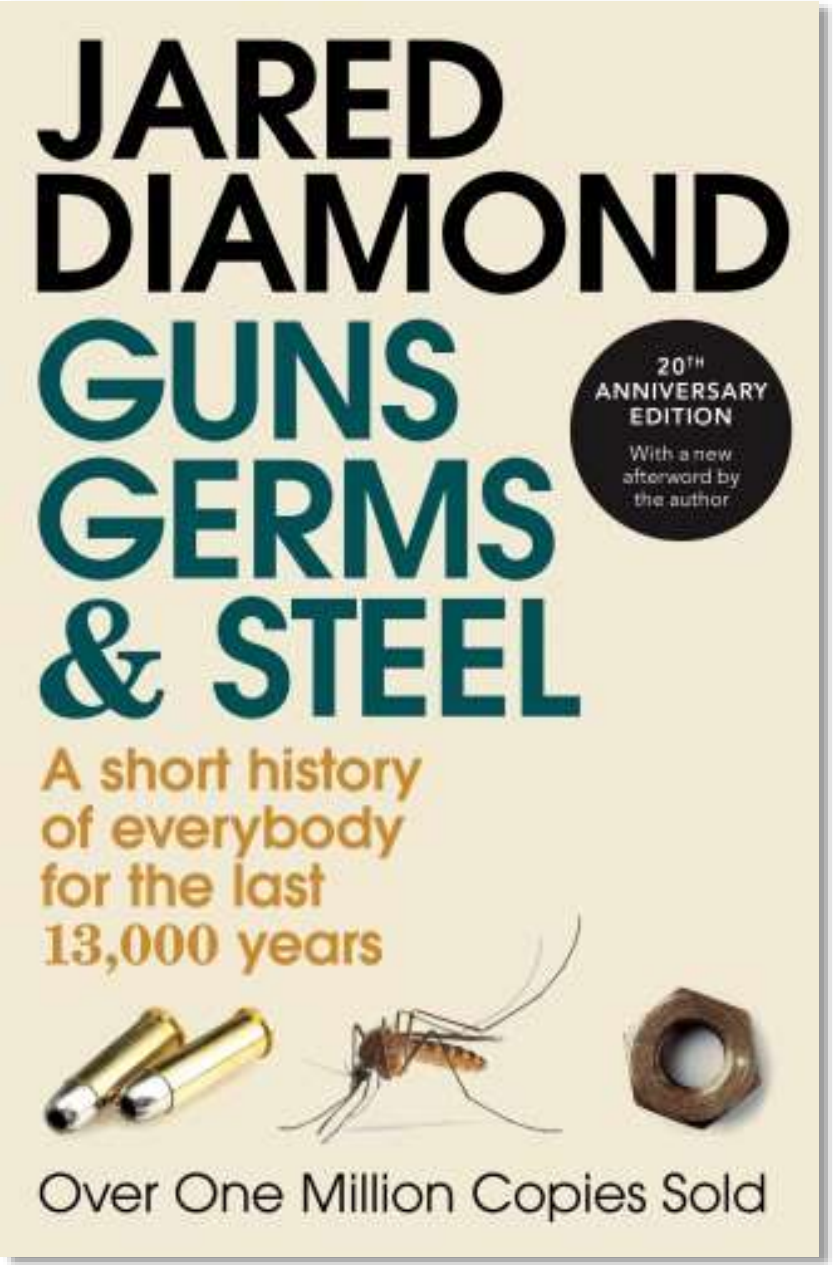
- Microservices are as much about the organisation as they are the tech
- Access to resources -- tools, knowledge, and time -- is vital for success
- Architecture is becoming more about technical leadership
- Continuous delivery is the catalyst to drive change

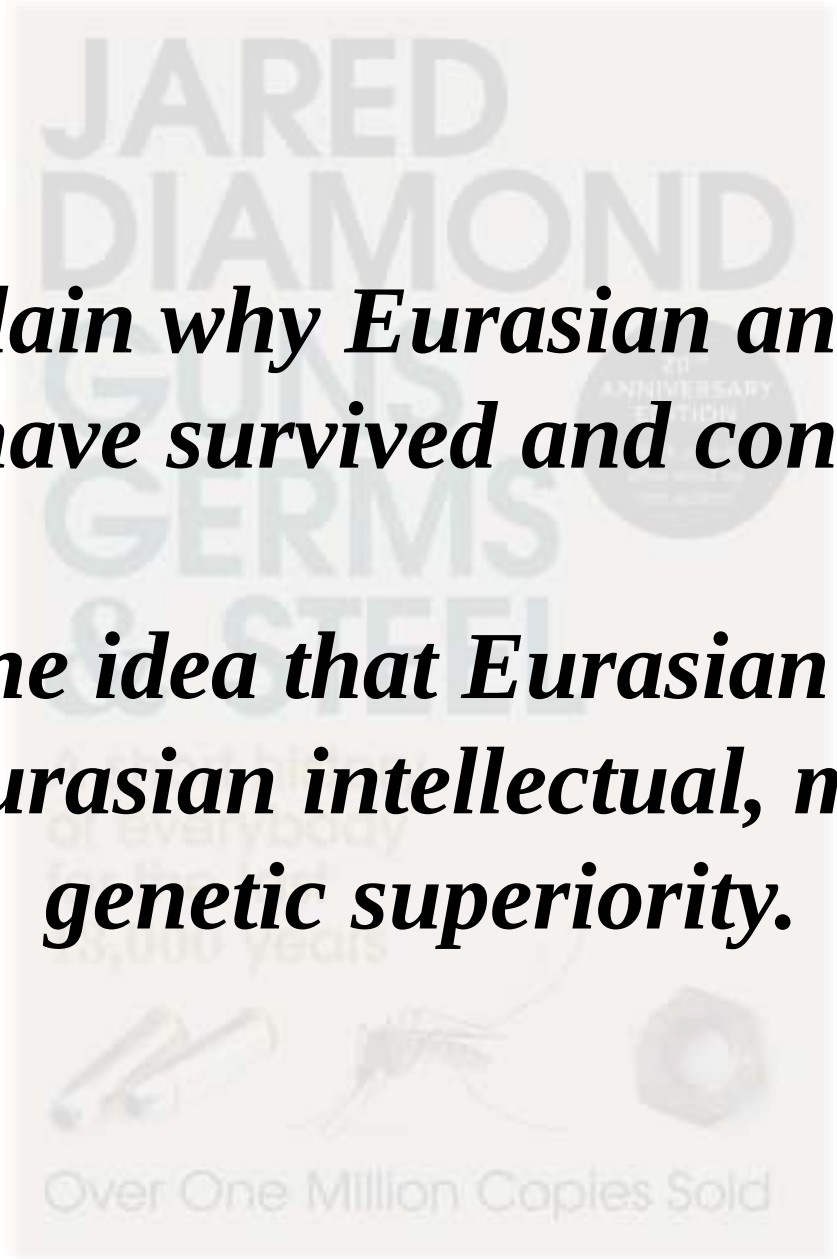
@danielbryantuk

- Independent Technical Consultant, CTO at SpectoLabs
  - Architecture, DevOps, Java, microservices, cloud, containers
  - Continuous Delivery (CI/CD) advocate
  - Leading change through technology and teams

bit.ly/2jWDSF7

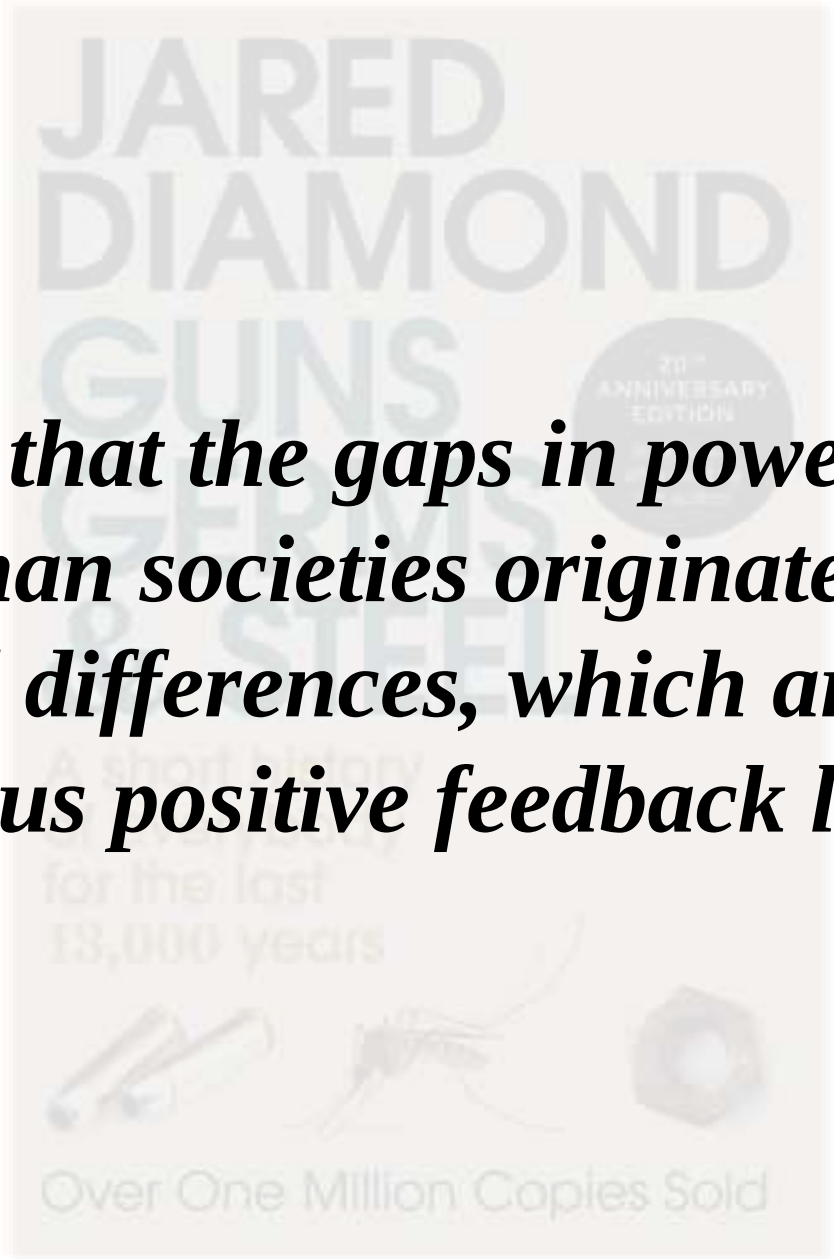
Setting the Scene


JARED
DIAMOND
GUNS
GERMS
& STEEL
20TH
ANNIVERSARY
EDITION
With a new
afterword by
the author
A short history
of everybody
for the last
13,000 years
Over One Million Copies Sold



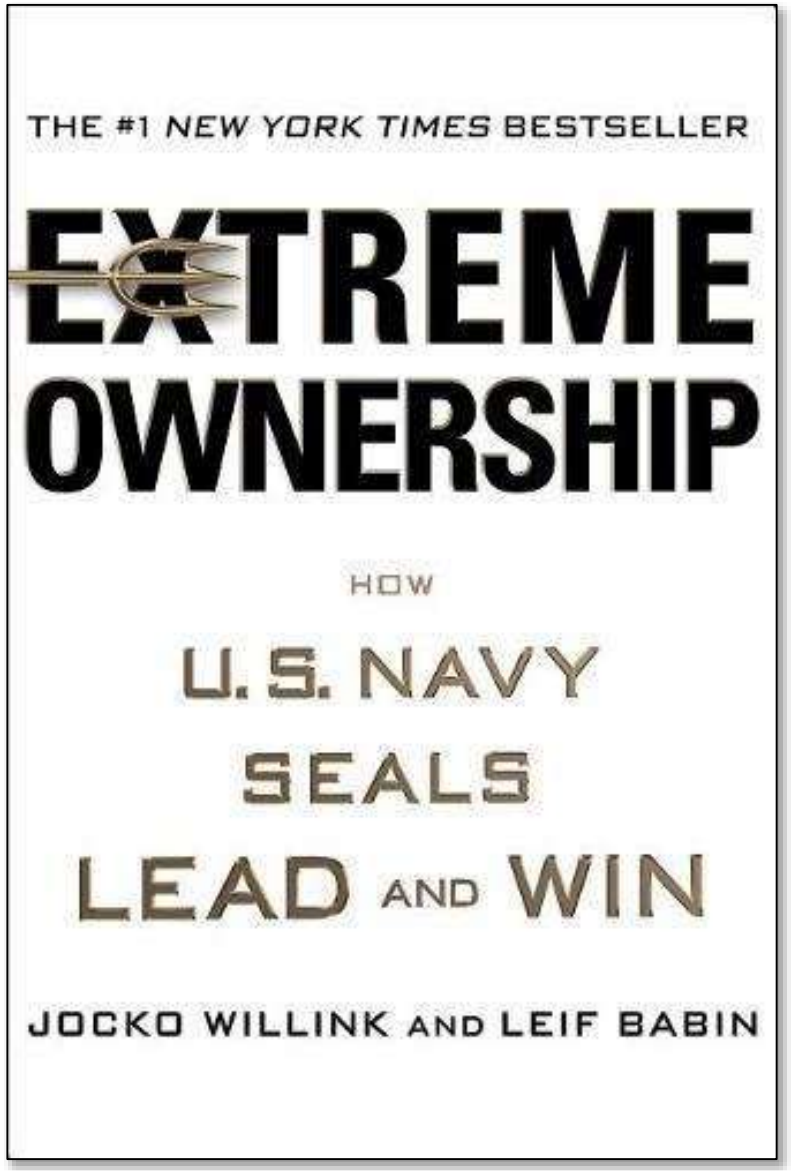
Attempts to explain why Eurasian and North African civilizations have survived and conquered others

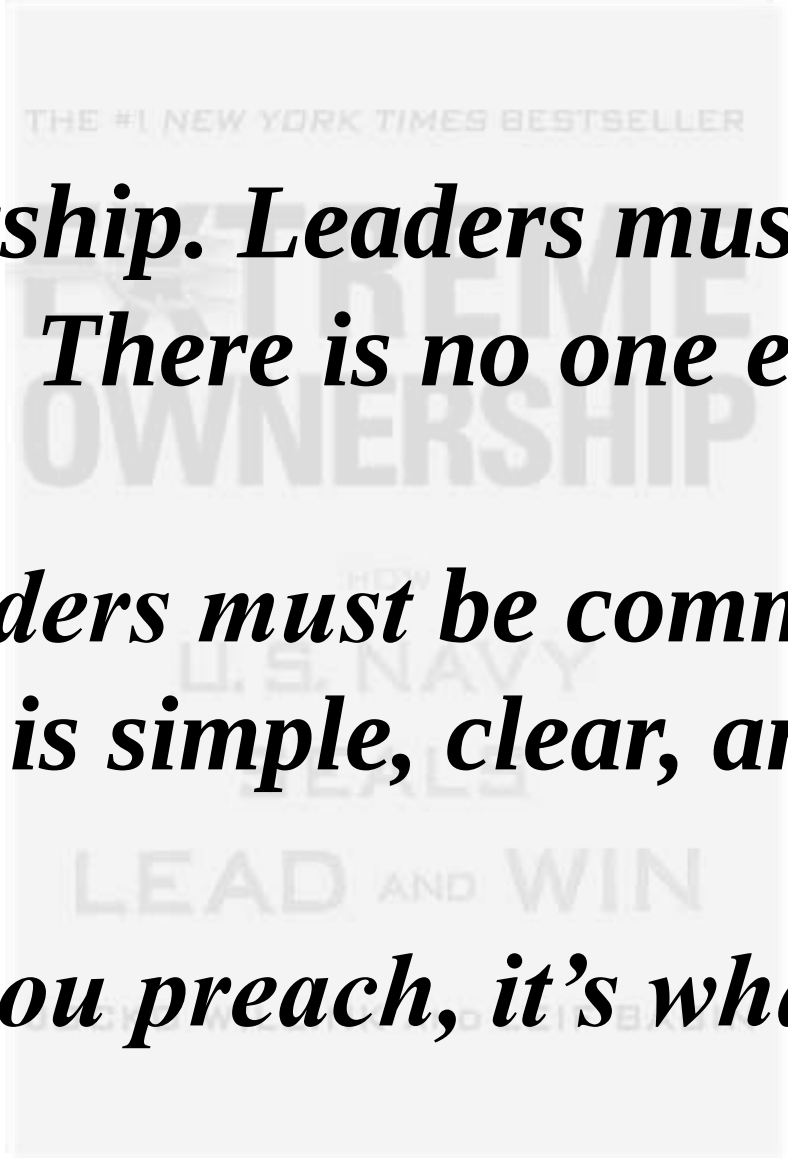
Argues against the idea that Eurasian hegemony is due to any form of Eurasian intellectual, moral, or inherent genetic superiority.



Diamond argues that the gaps in power and technology between human societies originate primarily in environmental differences, which are amplified by various positive feedback loops.

Diamond argues that the gaps in power and technology between human societies originate primarily in environmental differences, which are amplified by various positive feedback loops.

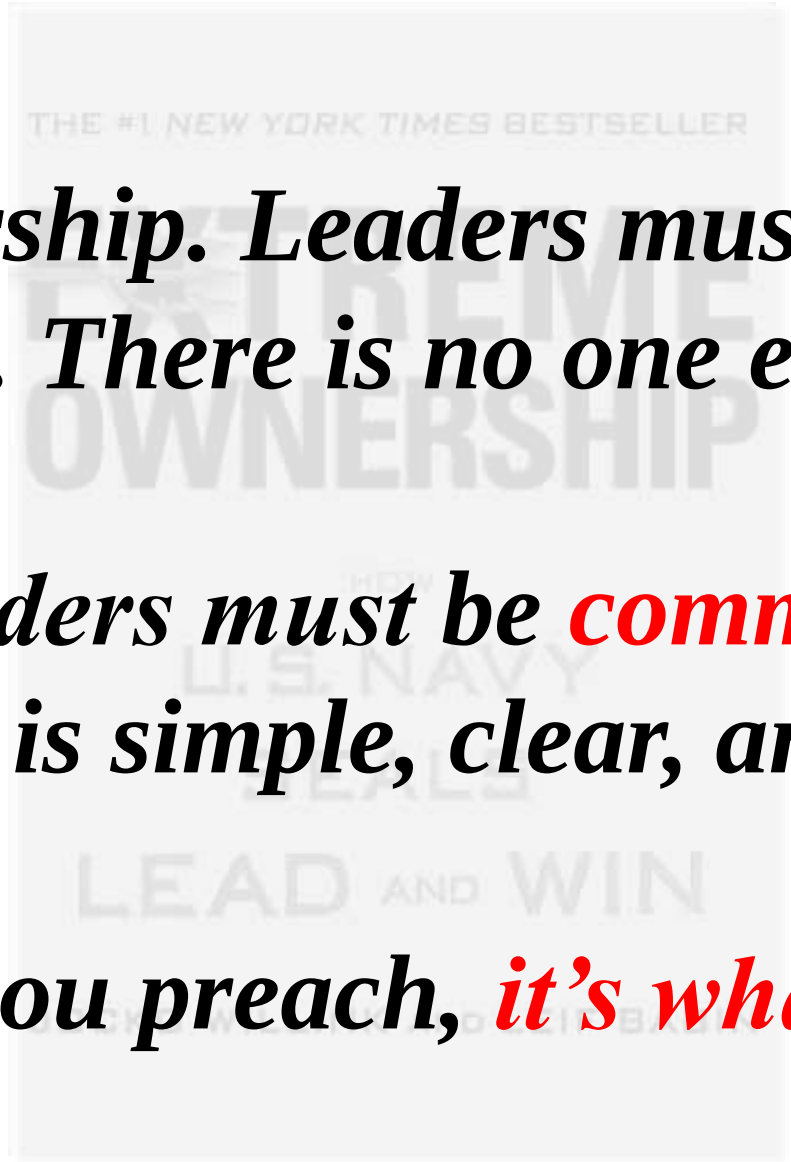

THE #1 NEW YORK TIMES BESTSELLER
EXTREME OWNERSHIP
HOW
U.S. NAVY SEALS LEAD AND WIN
JOCKO WILLINK AND LEIF BABIN



"Extreme Ownership. Leaders must own everything in their world. There is no one else to blame."

"Plans and orders must be communicated in a manner that is simple, clear, and concise. "

"It's not what you preach, it's what you tolerate"



"Extreme Ownership. Leaders must own everything in their world. There is no one else to blame."

"Plans and orders must be communicated in a manner that is simple, clear, and concise. "

"It's not what you preach, it's what you tolerate"

Environmental resources **versus** leadership

- OR -

Environmental resources **and** leadership

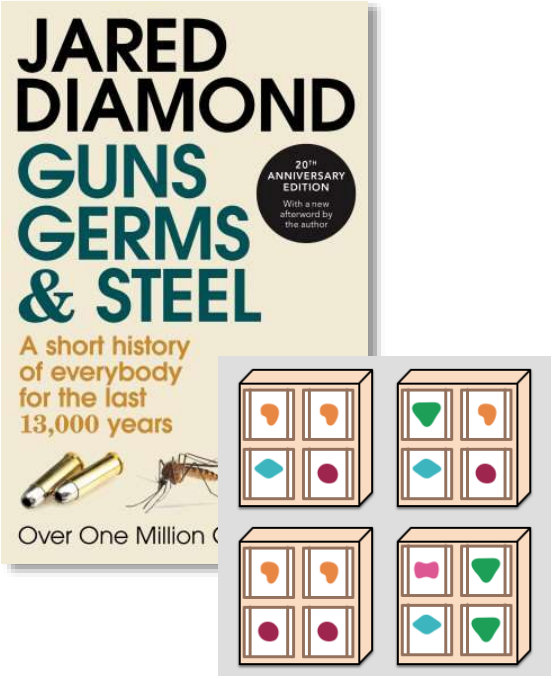
Creating a Surplus

Creating a surplus

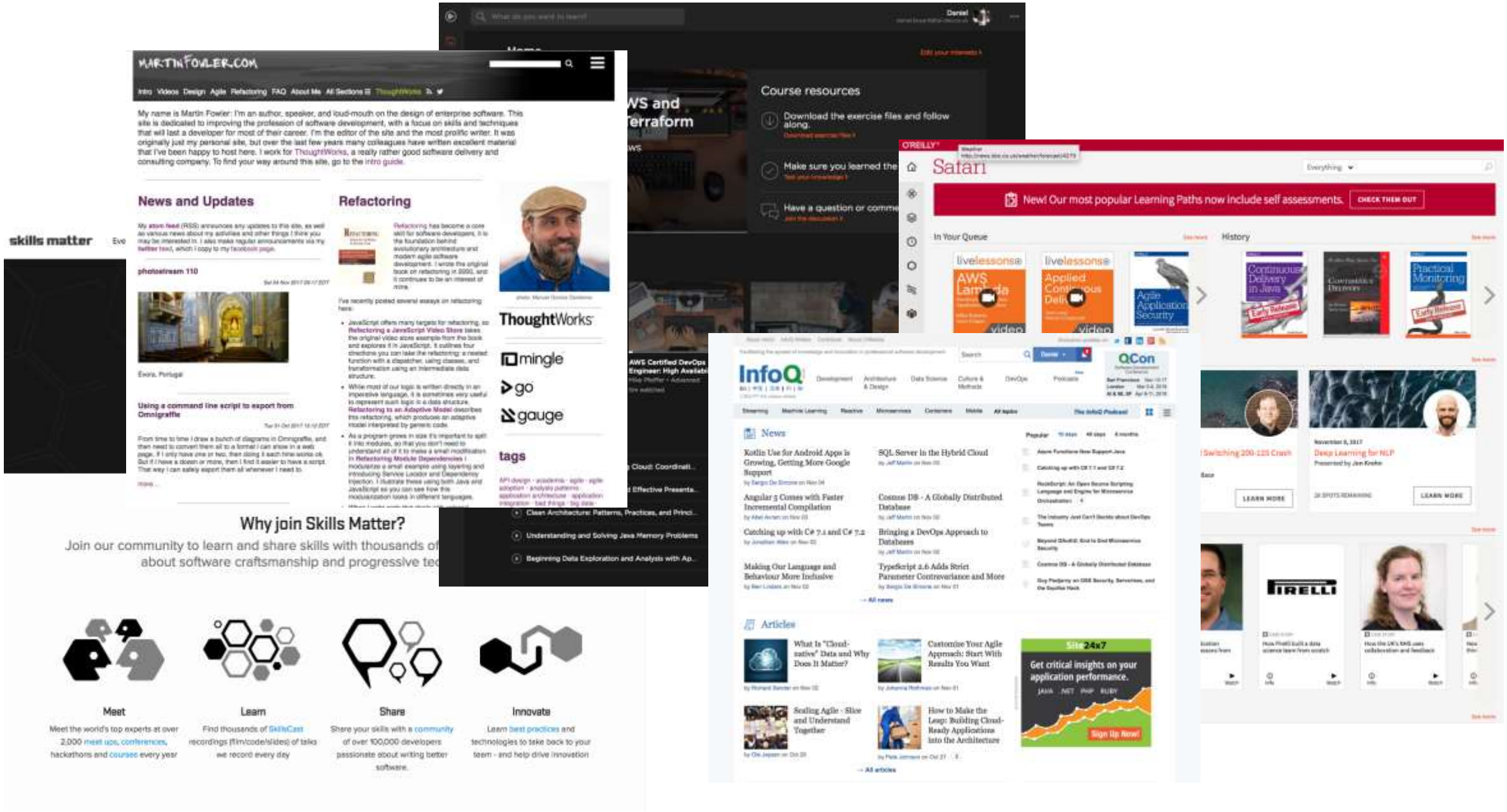
https://commons.wikimedia.org/w/index.php?curid=471913

- The fertile crescent had great availability to suitable plant and animal species for domestication
  - Eurasian grains were easier to sow and richer in protein
  - 13 large animals in Eurasia compared to 1 in America
- This led to a food surplus
- Allowed specialisation in society
- Fostered social and technological innovations

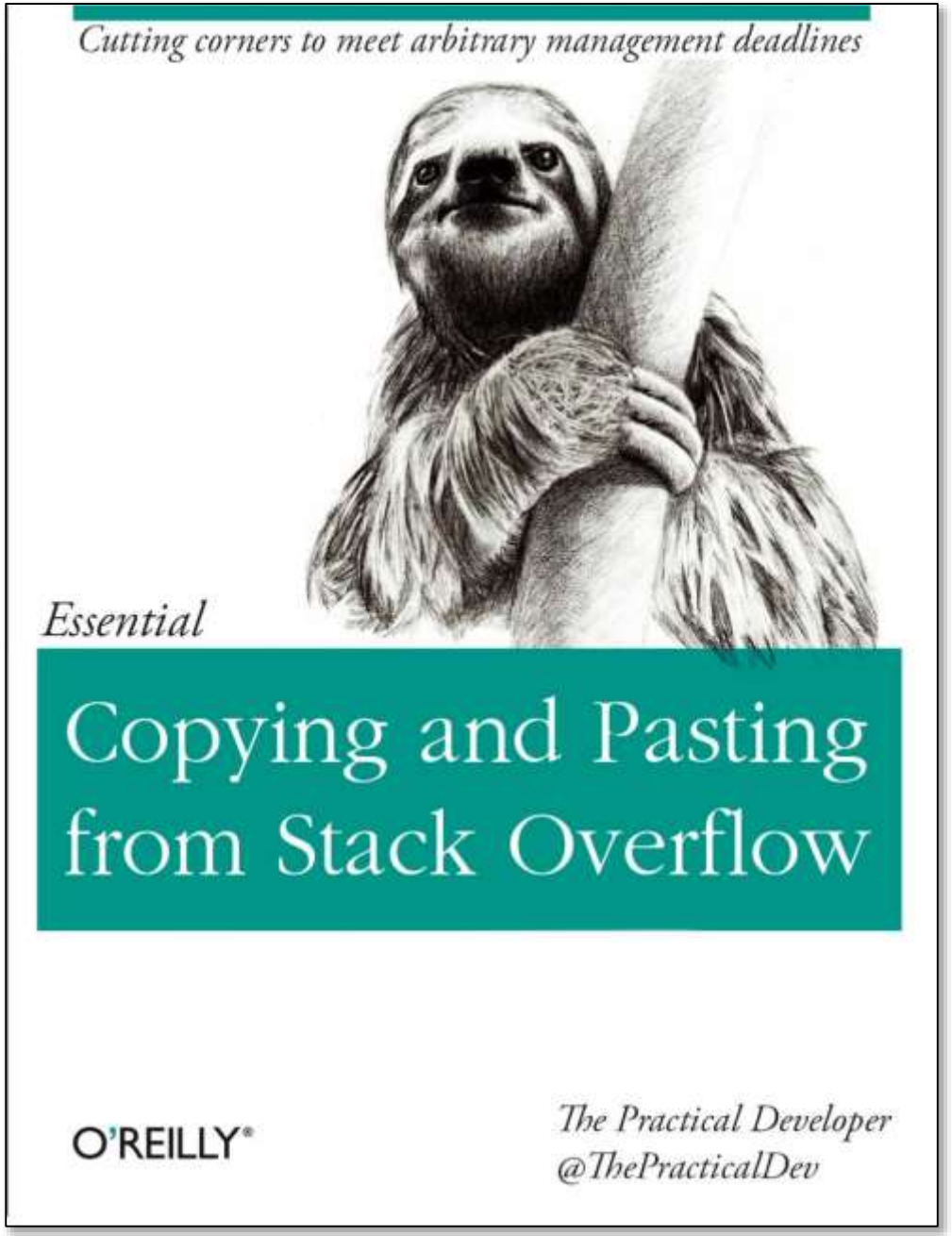
Creating a surplus



- For us as technologists, the resources are access to existing solutions, knowledge, and time
- Our “workhorses” are automation
- This leads to an technology surplus, specialisation, and progress
- …and ultimately, an “innovation” surplus


MARTINFOWLER.COM
News and Updates
Refactoring
ThoughtWorks
mingle
go
gauge
tags
skills matter
Course resources
Download the exercise files and follow along.
Make sure you learned the
Have a question or comme
Clean Architecture: Patterns, Practices, and Princi...
Understanding and Solving Java Memory Problems
Beginning Data Exploration and Analysis with Ap...
Why join Skills Matter?
Meet
Learn
Share
Innovate
Safari
New! Our most popular Learning Paths now include self assessments.
CHECK THEM OUT
In Your Queue
History
InfoQ
QCon
News
Articles
Get critical insights on your application performance.
Sign Up Now!
PIRELLI

ANTIPATTERN

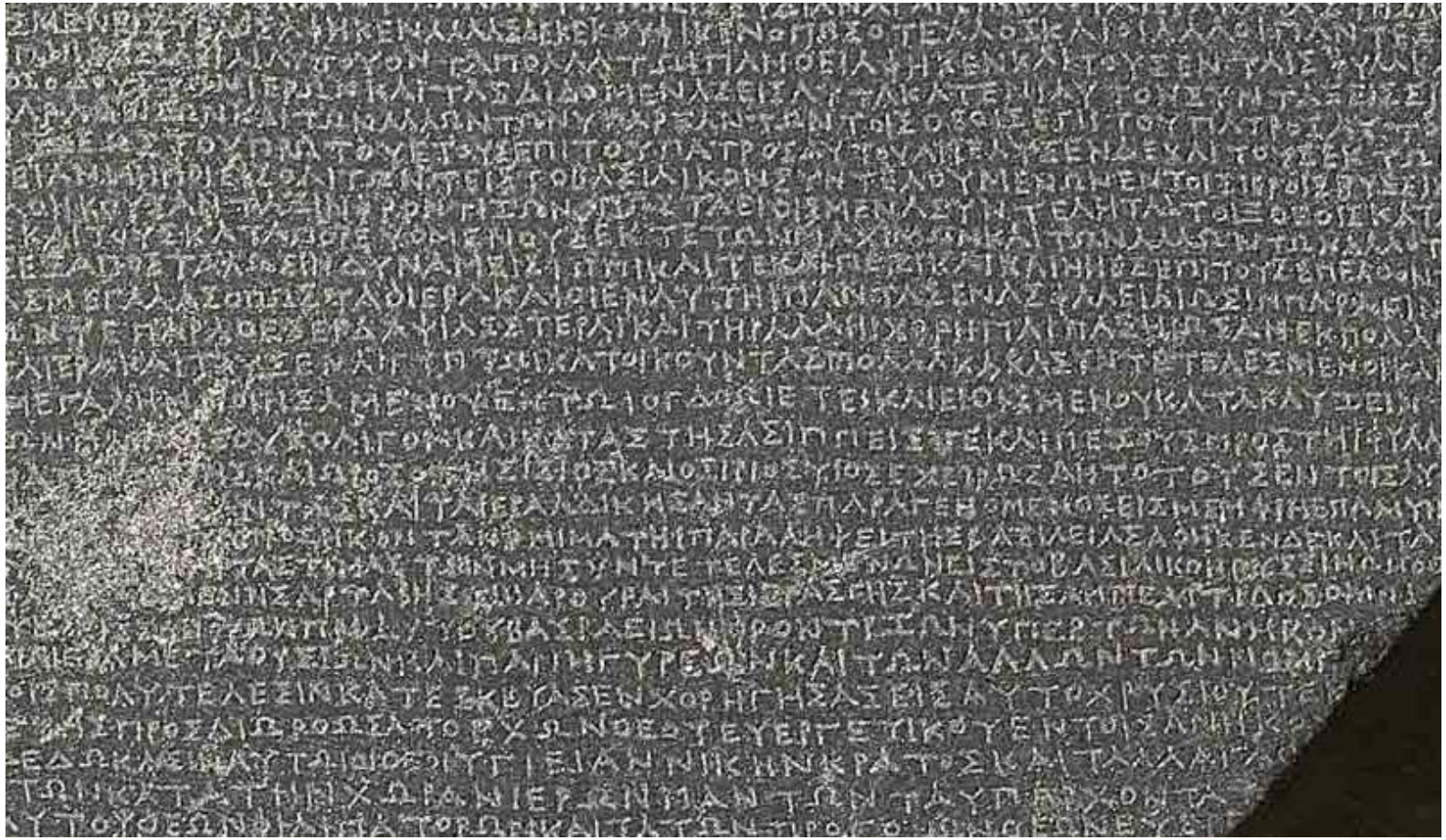


ANTIPATTERN

Conference-Driven Development

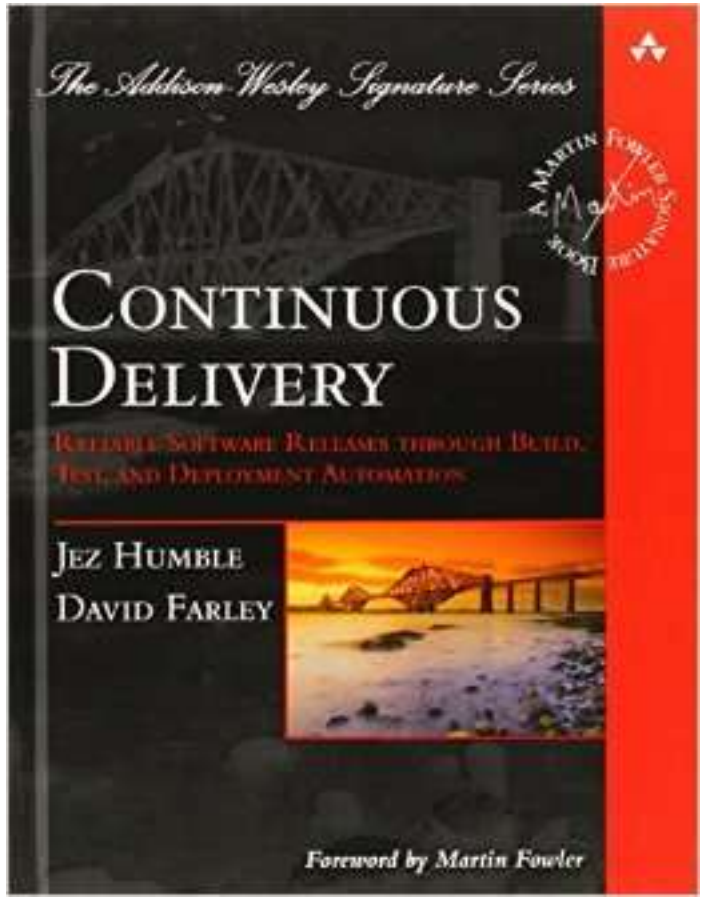
ANTIPATTERN

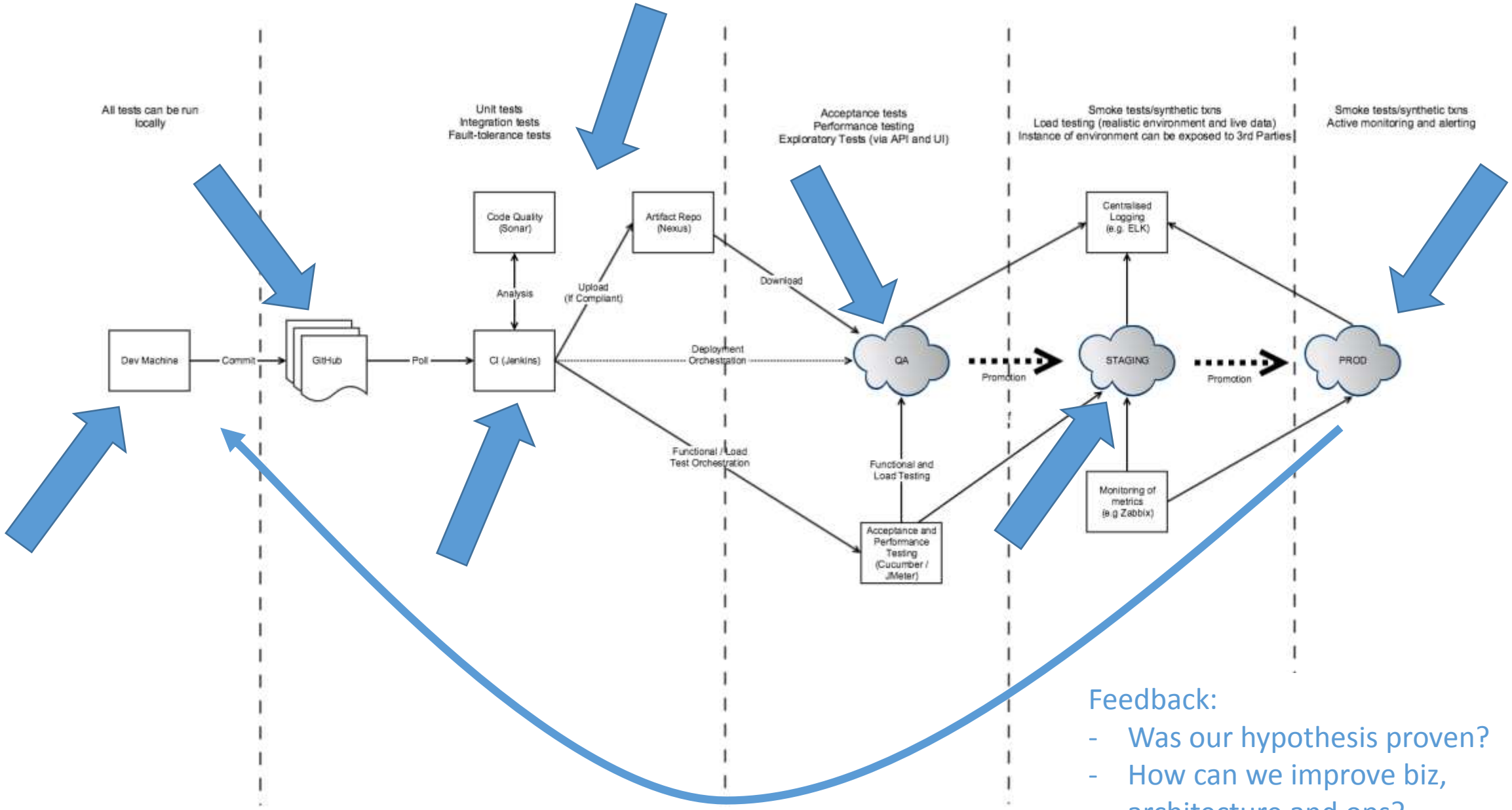
Lost in translation



Automation: Continuous Delivery

- Produce valuable and robust software in short cycles
- Optimising for feedback and learning
- Not (necessarily) Continuous Deployment

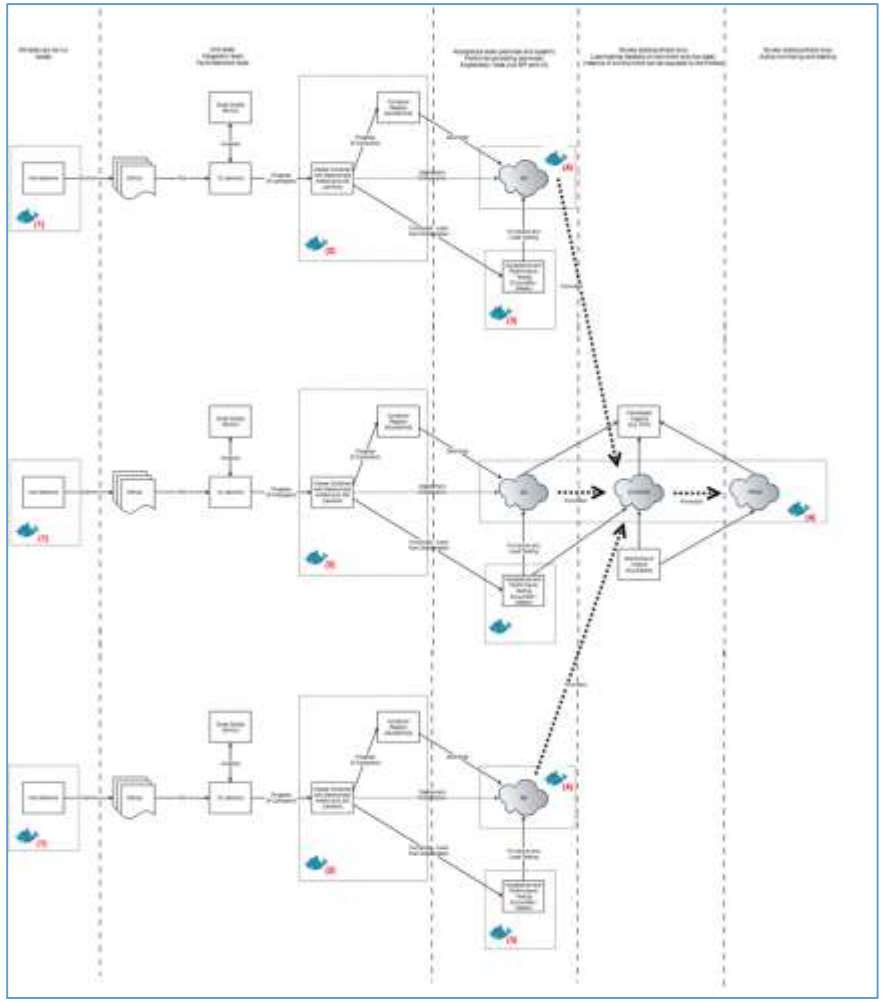


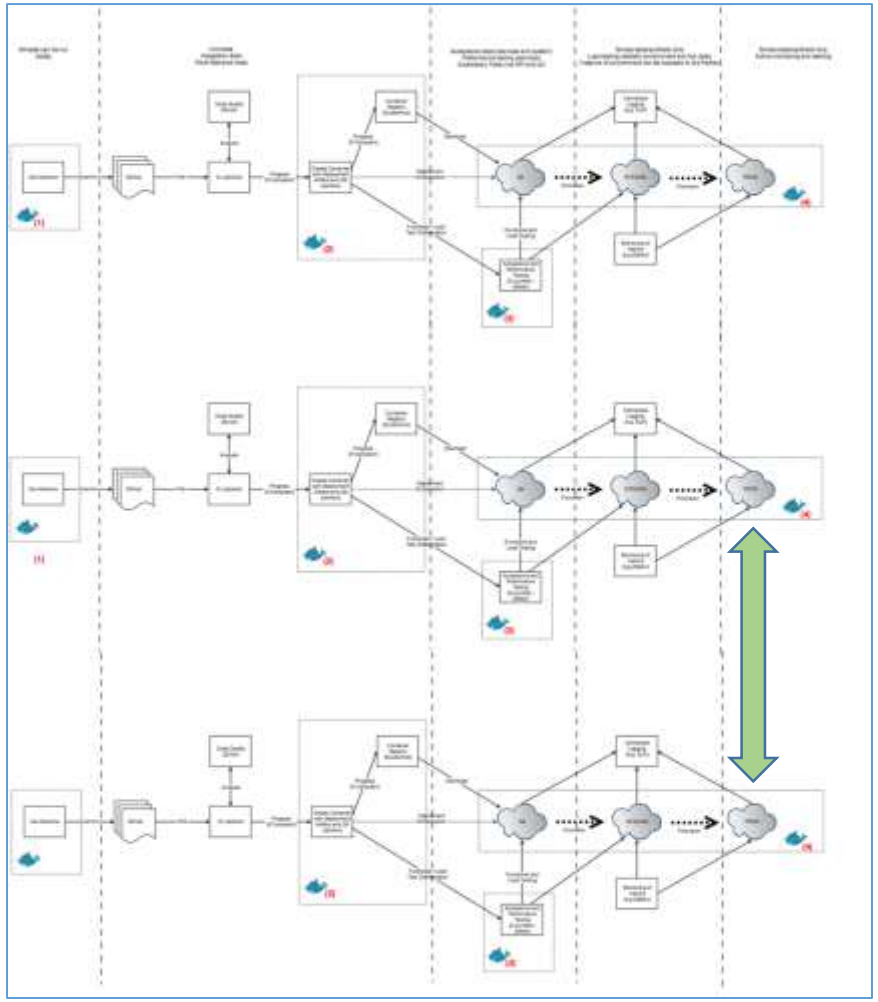


Feedback:

- Was our hypothesis proven?
- How can we improve biz, architecture and ops?

Microservices multiply the challenges





Talking of (service) contracts…

```java
import au.com.dius.pact.consumer.dsl.PactDslWithProvider;
import au.com.dius.pact.consumer.examplecliens.ConsumerClient;
import au.com.dius.pact.consumer.ConsumerPactTest;
import au.com.dius.pact.model.PactFragment;
import org.junit.Assert;

import java.io.IOException;
import java.util.HashMap;
import java.util.Map;

import static org.junit.Assert.assertEquals;

public class ExampleJavaConsumerPactTest extends ConsumerPactTestMk2 {

    @Override
    protected RequestResponsePact createFragment(PactDslWithProvider builder) {
        Map<String, String> headers = new HashMap<String, String>();
        headers.put("testreqheader", "testreqheadervalue");

        return builder
            .given("test state") // NOTE: Using provider states are optional, you can leave it out
            .uponReceiving("ExampleJavaConsumerPactTest test interaction")
                .path("/")
                .method("GET")
                .headers(headers)
            .willRespondWith()
                .status(200)
                .headers(headers)
                .body("{\"responsetest\": true, \"name\": \"harry\"}")
            .given("test state 2") // NOTE: Using provider states are optional, you can leave it out
            .uponReceiving("ExampleJavaConsumerPactTest second test interaction")
                .method("OPTIONS")
                .headers(headers)
                .path("/second")
                .body("")
            .willRespondWith()
                .status(200)
                .headers(headers)
                .body("")
            .toPact();
    }
```

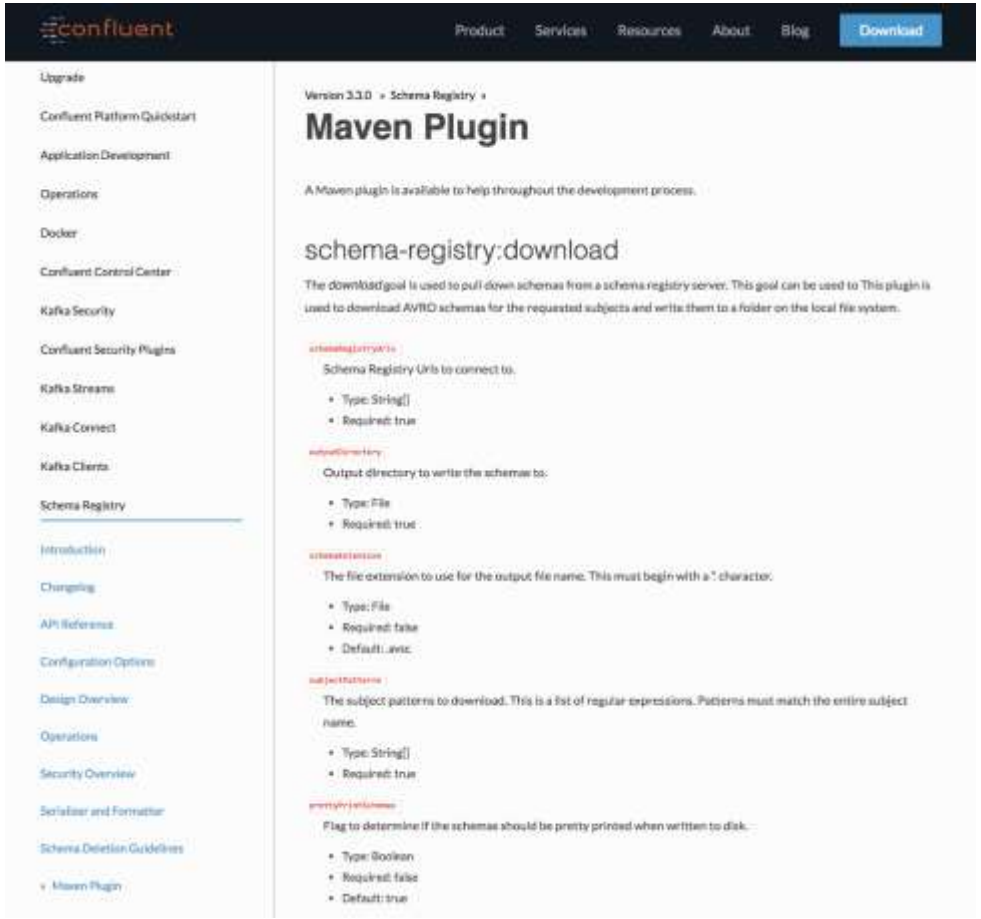
```
spring-cloud-contract-samples / producer / src / test / resources / contracts / beer / intoxication / 3_drunk.groovy

marcingrzejszczak Added tutorial for scenarios

1 contributor

37 lines (33 sloc) | 520 Bytes
```

```
package contracts.beer.intoxication

import org.springframework.cloud.contract.spec.Contract

Contract.make {
        description("""
Represents last step of getting fully drunk

given:
        you were drunk
when:
        you get a beer
then:
        you'll be wasted
""")
        request {
                method 'POST'
                url '/beer'
                body(
                        name: "marcin"
                )
                headers {
                        contentType(applicationJson())
                }
        }
        response {
                status 200
                body(
                        previousStatus: "DRUNK",
                        currentStatus: "WASTED"
                )
                headers {
                        contentType(applicationJson())
                }
        }
}
```

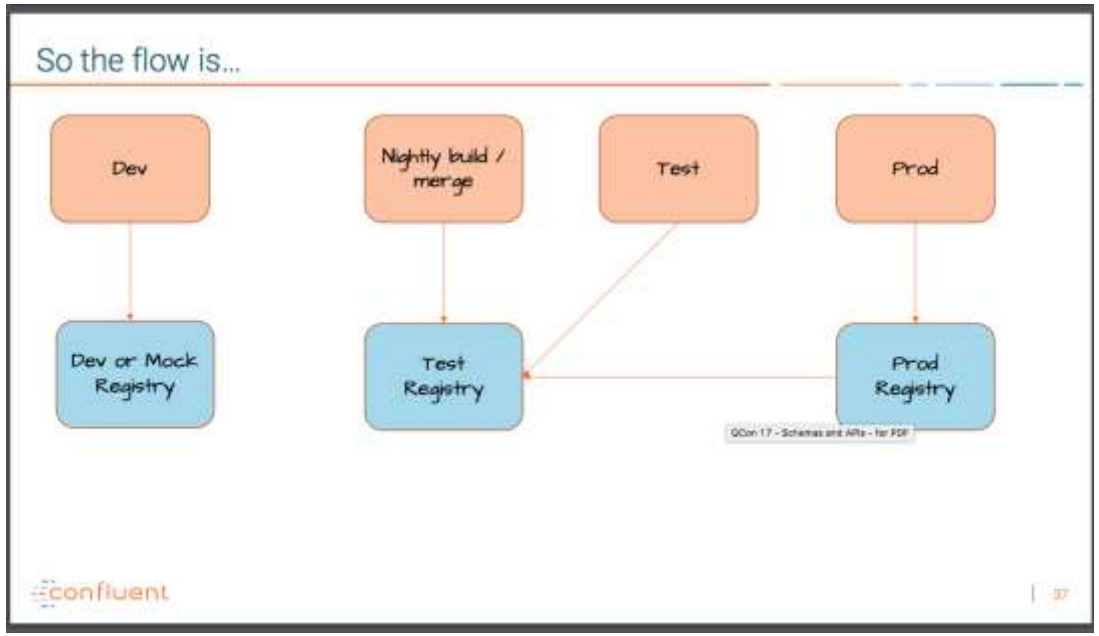
docs.pact.io

cloud.spring.io/spring-cloud-contract
github.com/spring-cloud-samples/spring-cloud-contract-samples

Talking of (messaging) contracts…



docs.confluent.io/current/schema-registry/docs/maven-plugin.html



www.infoq.com/presentations/contracts-streaming-microservices

Velocity (with stability) is key to business success

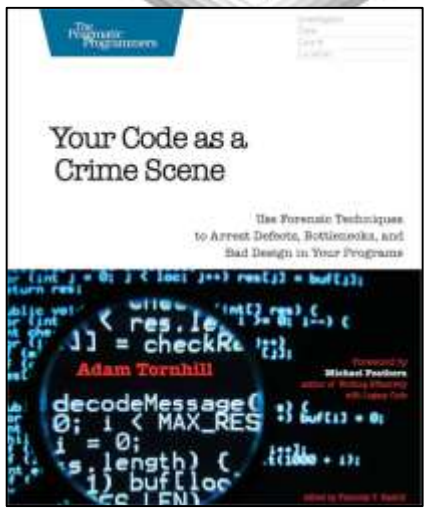
"Continuous delivery is achieved when stability and speed can satisfy business demand.

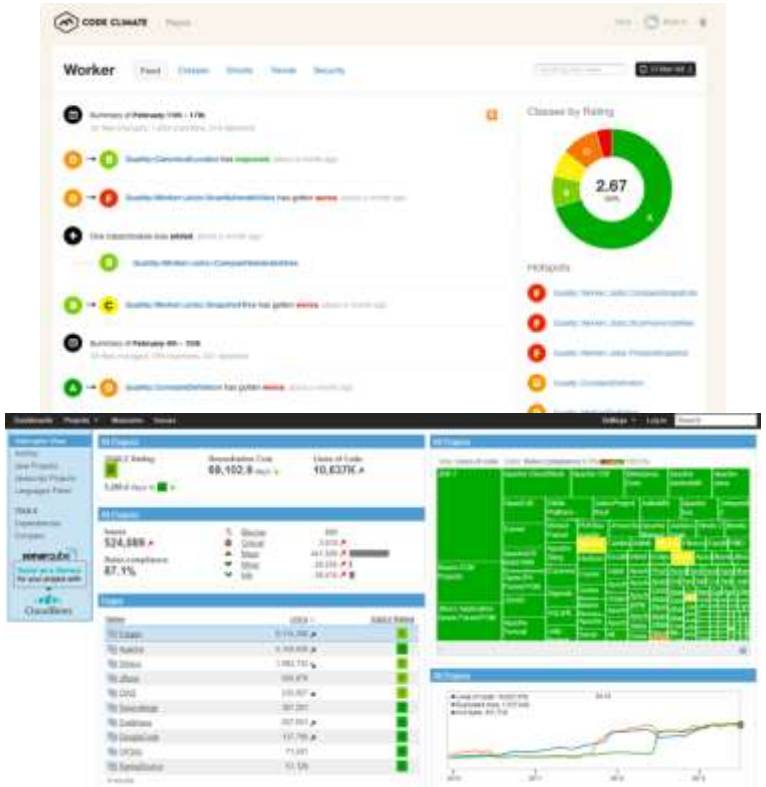
Discontinuous delivery occurs when stability and speed are insufficient."

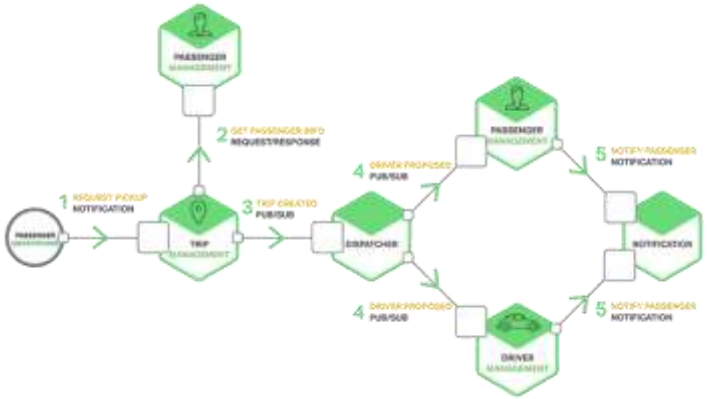
- Steve Smith (@SteveSmithCD)

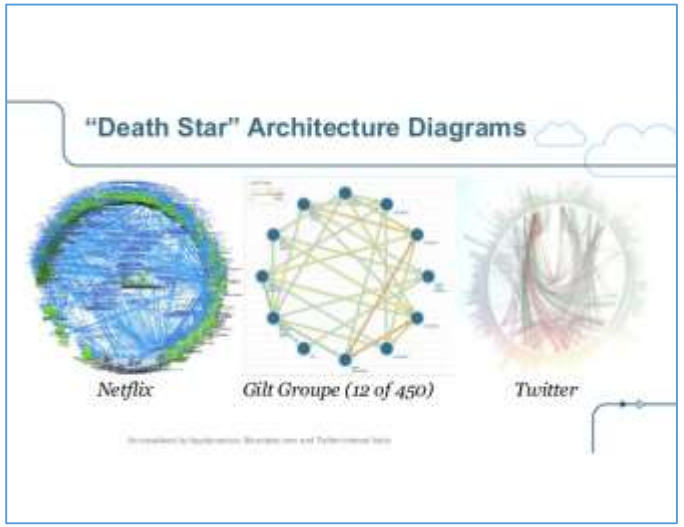
Visibility for the business

Architectural feedback





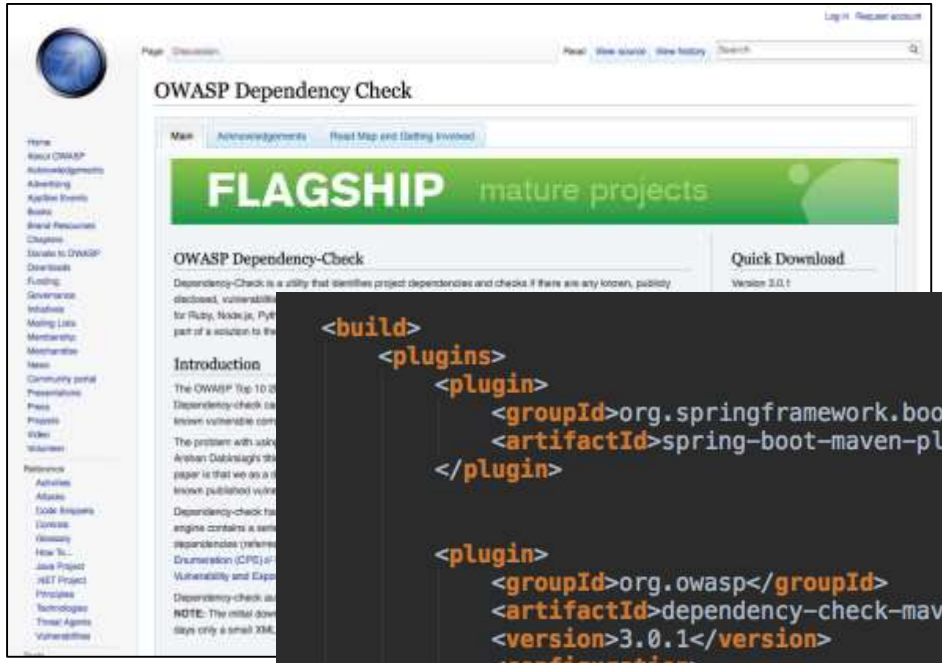




Operational visibility

- Logging
  - The 10 Commandments of Logging
  - The Log: What every software engineer should know
- Monitoring and alerting
  - Rob Ewaschuk's Philosophy on Alerting”
  - Brendan Gregg's USE method

www.owasp.org/index.php/OWASP_Dependency_Check




```
<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactI
        </plugin>

        <plugin>
            <groupId>org.owasp</groupId>
            <artifactId>dependency-check-maven</artifactId>
            <version>3.0.1</version>
            <configuration>
                <failBuildOnCVSS>8</failBuildOnCVSS>
            </configuration>
            <executions>
                <execution>
                    <goals>
                        <goal>check</goal>
                    </goals>
                </execution>
            </executions>
        </plugin>
    </plugins>

</build>
```

```
[INFO] Processing Complete for NVD CVE - 2017 (43421 ms)
[INFO] Processing Complete for NVD CVE - 2008 (44885 ms)
[INFO] Processing Complete for NVD CVE - 2010 (40114 ms)
[INFO] Processing Complete for NVD CVE - 2011 (39776 ms)
[INFO] Processing Complete for NVD CVE - 2016 (45011 ms)
[INFO] Processing Complete for NVD CVE - 2014 (41809 ms)
[INFO] Begin database maintenance.
[INFO] End database maintenance.
[INFO] Check for updates complete (91508 ms)
[INFO] Analysis Started
[INFO] Finished Archive Analyzer (0 seconds)
[INFO] Finished File Name Analyzer (0 seconds)
[INFO] Finished Jar Analyzer (1 seconds)
[INFO] Finished Central Analyzer (7 seconds)
[INFO] Finished Dependency Merging Analyzer (0 seconds)
[INFO] Finished Version Filter Analyzer (0 seconds)
[INFO] Finished Hint Analyzer (0 seconds)
[INFO] Created CPE Index (1 seconds)
[INFO] Finished CPE Analyzer (2 seconds)
[INFO] Finished False Positive Analyzer (0 seconds)
[INFO] Finished Cpe Suppression Analyzer (0 seconds)
[INFO] Finished NVD CVE Analyzer (0 seconds)
[INFO] Finished Vulnerability Suppression Analyzer (0 seconds)
[INFO] Finished Dependency Bundling Analyzer (0 seconds)
[INFO] Analysis Complete ([illegible] seconds)
[WARNING]

[illegible] or more dependencies were identified with known vulnerabili

jersey-apache-client4-1.19.1.jar (cpe:/a:oracle:oracle_client:1
xstream-1.4.9.jar (cpe:/a:x-stream:xstream:1.4.9, cpe:/a:xstrea
netty-codec-4.0.27.Final.jar (cpe:/a:netty_project:netty:4.0.27
ognl-3.0.8.jar (ognl:ognl:3.0.8, cpe:/a:ognl_project:ognl:3.0.8
groovy-2.4.12.jar (cpe:/a:apache:groovy:2.4.12, org.codehaus.gr

See the dependency-check report for more details.
```

ANTIPATTERN

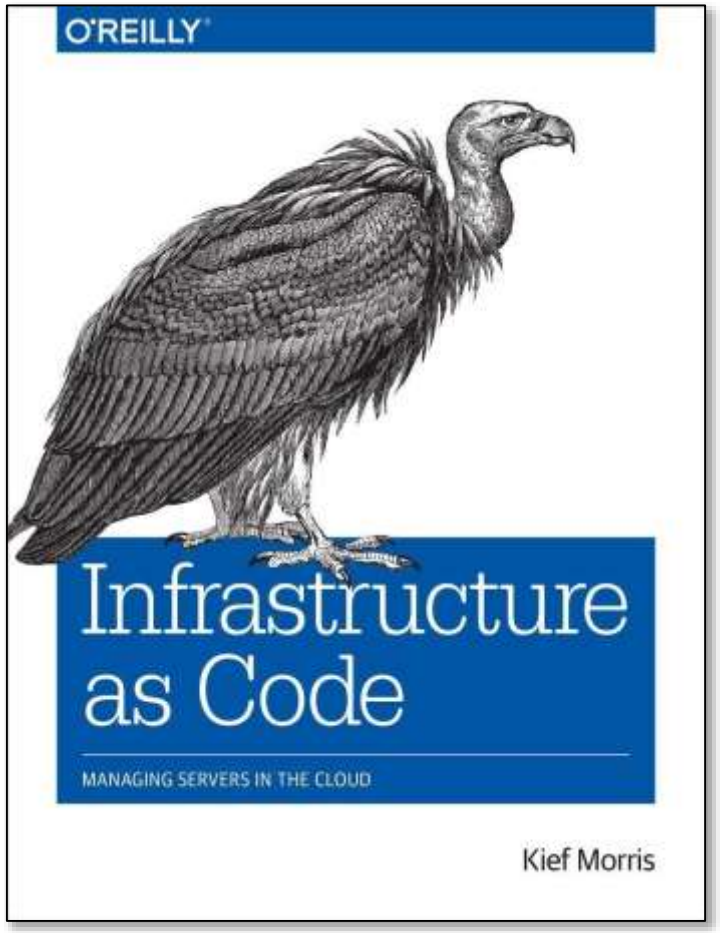
Automation: Should I build my own platform?

Probably not

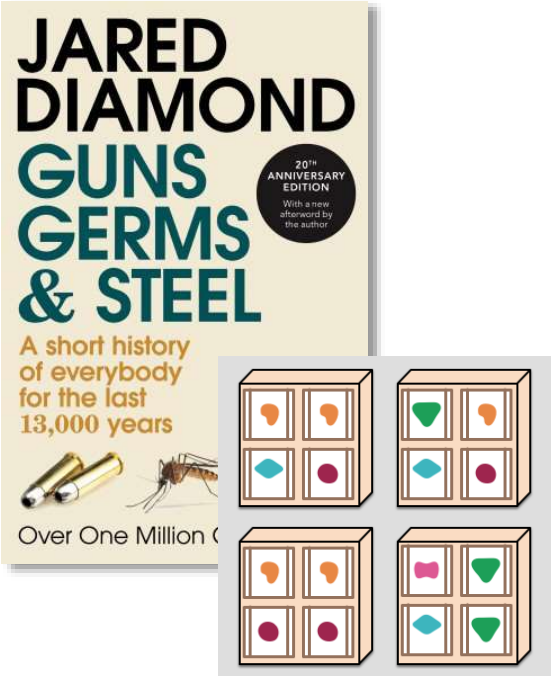
(Unless you are Google, AWS or IBM)

Whatever you decide…

push it through a pipeline ASAP!



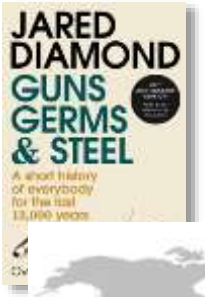
Creating a surplus

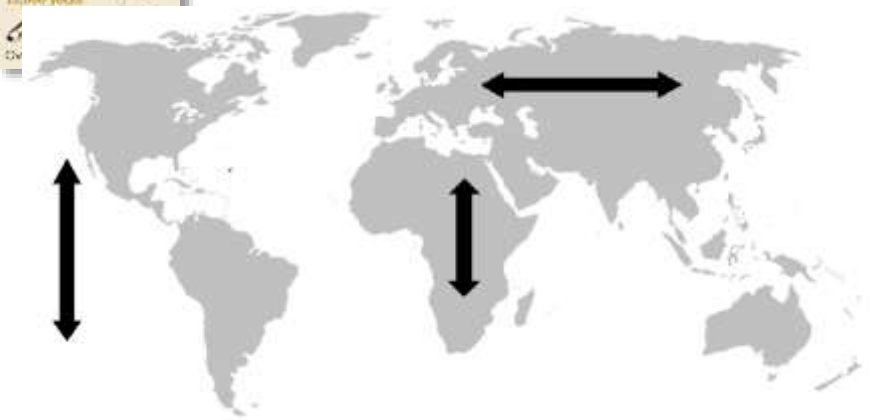


- Access to existing solutions, knowledge, and time
  - Share and standardise
- Automation is essential for microservices
- Bake-in metrics to your services (and platform) to enable feedback

30,00ft View: Continental Scale

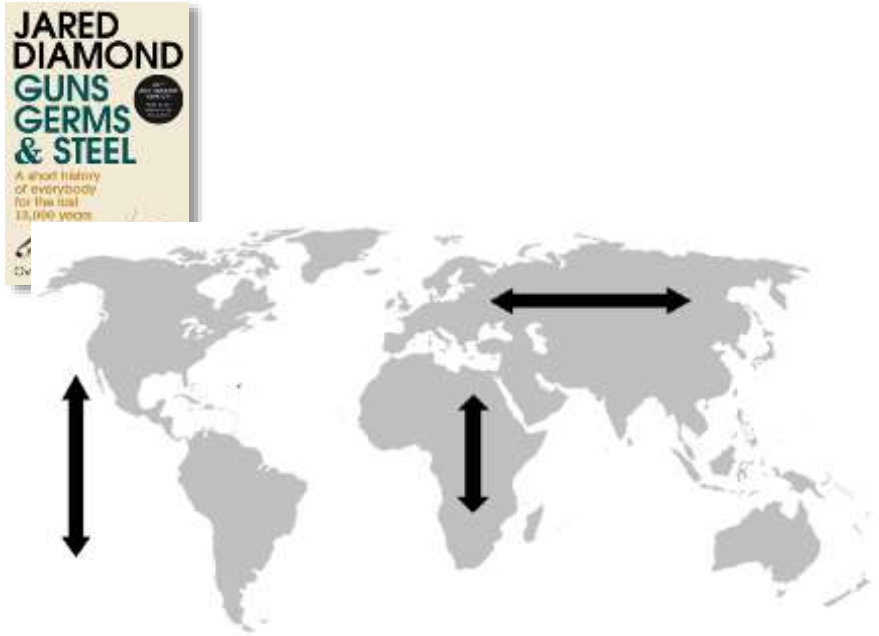
Continental scale: Spreading success





- Continental axis for Eurasia is east-west
- Africa and America: north-south
- East-west facilitated the spread of crops, innovation and disease (immunity) through latitude, climate, and ease of travel

Continental scale: Spreading success

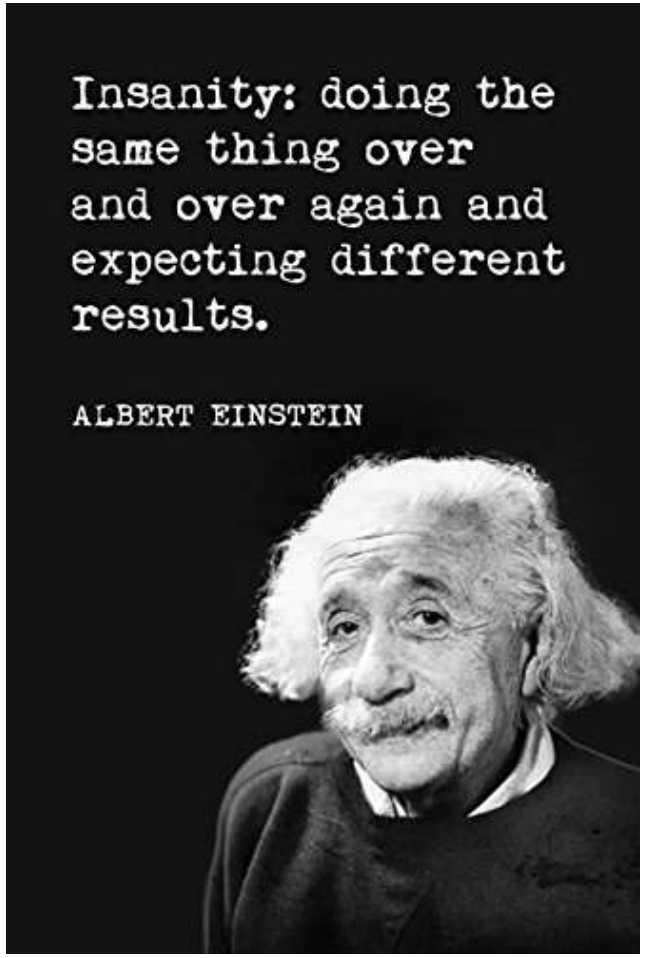


- Standardised technology
  - With microservices it's about the platforms, protocols and deployment
  - CNCF, Kubernetes, Cloud Foundry, gRPC, Kafka, service meshes etc.
- Defining social norms
  - Psychological safety
  - Immunised to "failure" (experimentation)

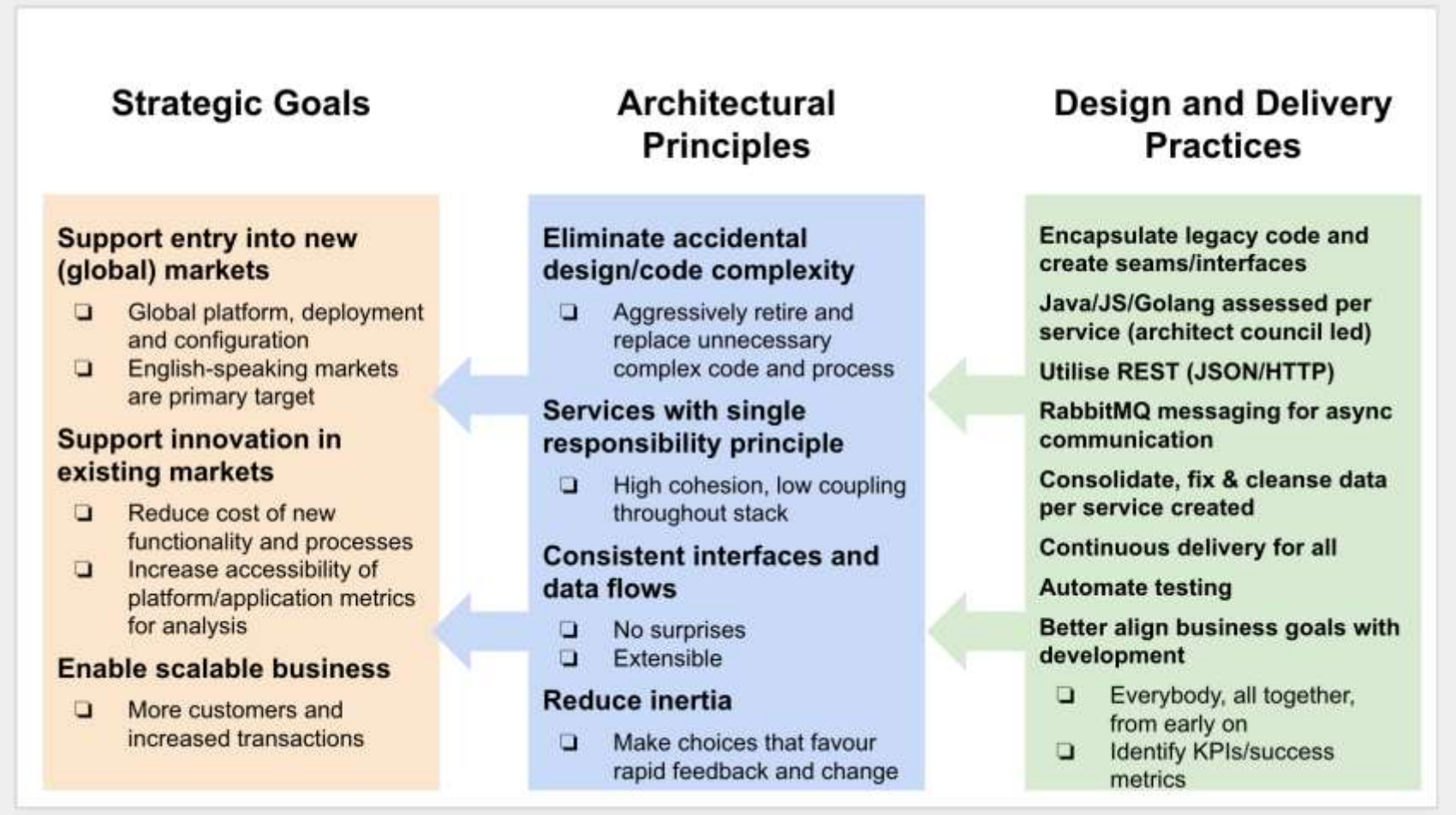
Sociotechnical insanity

ANTIPATTERN

- We created a mess with a monolith...
- But no change required with our approach to when implementing microservices???



Communicate the vision



Tactics: Technical Leadership is vital

- Promote shared understanding
  - Communication (bit.ly/1Ia3u8o)
- Risk management
  - Innovation tokens
- ‘Just enough’ up-front design
  - Boundaries, testing and 'glue'

Tactics: Technical Leadership is vital

- Conway's law is well accepted
  - Align teams with service (vice versa)
- Not so clear where 'Architects' sit
  - Overarching, consulting, or team?
- Pair product managers with tech leads for each team

Teams and technology: Innersource

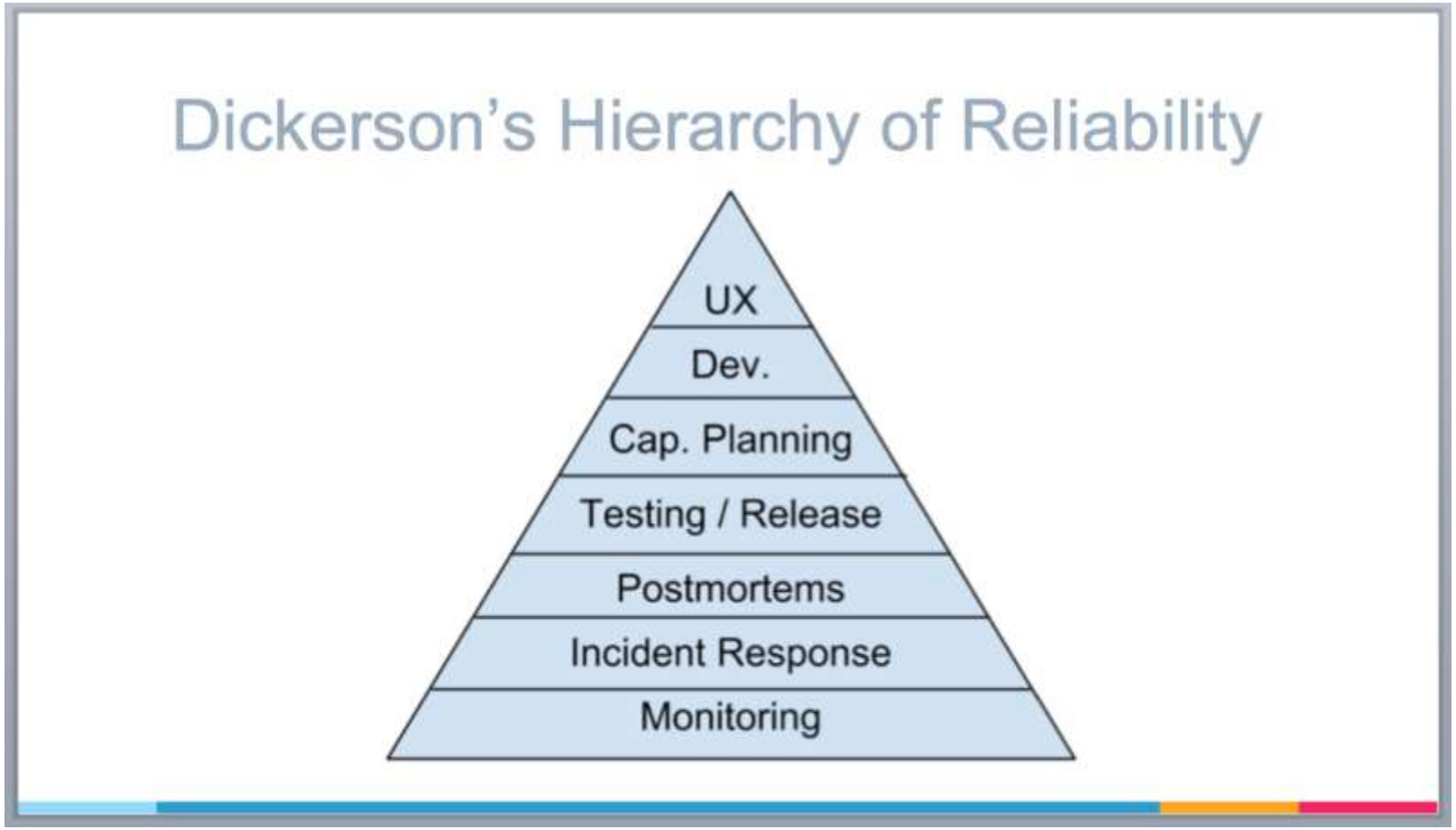
paypal

- Programmers share their work with a wide audience, instead of just with a manager or team. In most open source projects, anyone in the world is free to view the code, comment on it, learn new skills by examining it, and submit changes that they think will improve it or customize it to their needs.
- New code repositories (branches) based on the project can be made freely, so that sites with unanticipated uses for the code can adapt it. There are usually rules and technical support for re-merging different branches into the original master branch.
- People at large geographical distances, at separate times, can work on the same code or contribute different files of code to the same project.
- Communication tends to be written and posted to public sites instead of shared informally by word of mouth, which provides a history of the project as well as learning opportunities for new project members.
- Writing unit tests becomes a key programming task. a "unit test" is a small test that checks for a particular, isolated behavior such as rejecting incorrect input or taking the proper branch under certain conditions. In open source and inner source, testing is done constantly as changes are checked in, to protect against failures during production runs.

ersource.csp

When bad things happen, people are always involved…

Mikey Dickerson's Hierarchy of Reliability



www.infoq.com/news/2015/06/too-big-to-fail

Psychological Safety

- Shared belief that the team is safe for interpersonal risk taking
- Being able to show and employ one's self without fear of negative consequences
- Team members feel accepted and respected

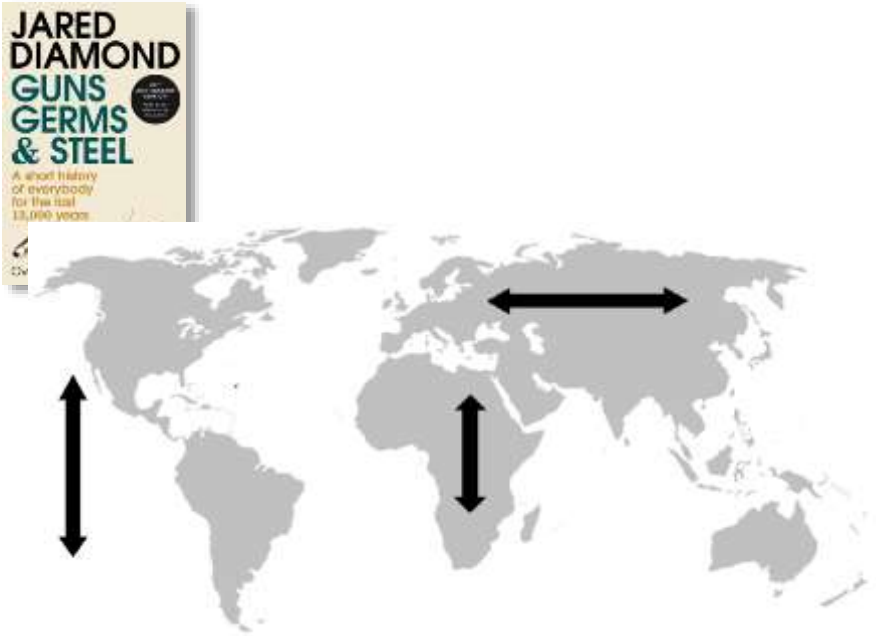
https://www.nytimes.com/2016/02/28/magazine/what-google-learned-from-its-quest-to-build-the-perfect-team.html

https://rework.withgoogle.com/blog/five-keys-to-a-successful-google-team/

What makes a Google team effective?

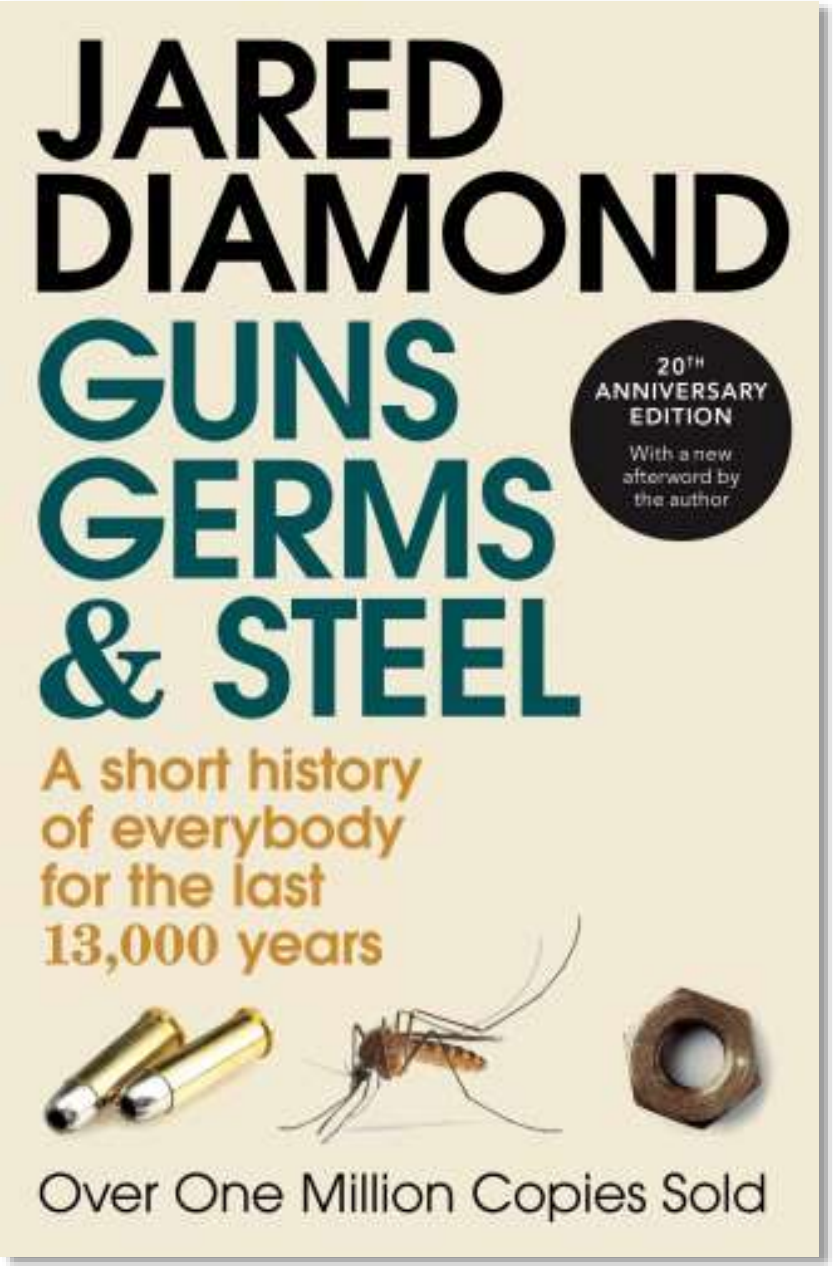
"When Rozovsky and her Google colleagues encountered the concept of psychological safety in academic papers, it was as if everything suddenly fell into place."

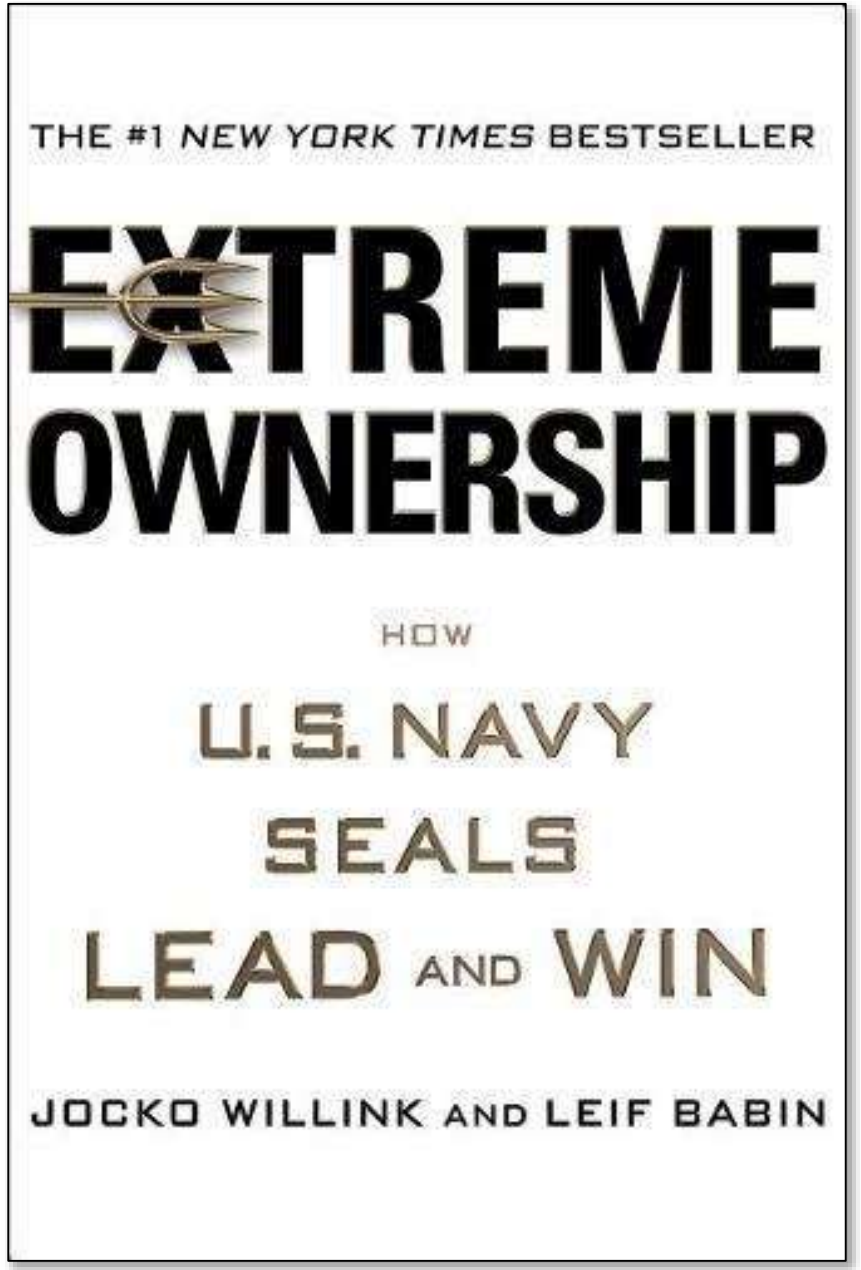
Continental scale: Spreading success



- Standardised technology
  - Strong (architectural) leadership needed
  - Think and communicate big picture
  - Platforms, protocols and deployment
- Defining social norms
  - Psychological safety
  - Shared understanding, balancing tech and product, game days etc

Wrapping up


JARED DIAMOND
GUNS GERMS & STEEL
20TH ANNIVERSARY EDITION
With a new afterword by the author
A short history of everybody for the last 13,000 years
Over One Million Copies Sold


THE #1 NEW YORK TIMES BESTSELLER
EXTREME OWNERSHIP
HOW
U.S. NAVY SEALS LEAD AND WIN
JOCKO WILLINK AND LEIF BABIN

In conclusion…

- Microservices are as much about the organisation as they are the tech
  - Recognise this, share the vision, and establish desired culture
- Access to resources -- tools, knowledge, and time -- is vital for success
  - Be selective, discuss ideas, capture high-level concept to share understanding
- Architecture is becoming more about technical leadership
  - Psychological safety is just as important and system design
  - Continuous delivery pipelines codify functional and nonfunctional requirements

Bedtime reading…

SOFTWARE BLOG

ABOUT ALL POSTS PRESENTATIONS TAGS

Christian Posta

Chief Architect, cloud application development @ Red Hat, author Microservices for Java Developers, open source enthusiast, committer @ Apache, Cloud, Integration, Kubernetes, Docker, OpenShift, Fabric8, #blogger

- Twitter
- Google+
- LinkedIn
- Github
- Stackoverflow

Low-risk Monolith to Microservice Evolution Part III

In the first part (part I) we introduced a strategy to bring microservices to our architecture without disrupting the current request flows and business value by looking at a concrete example. In the second part, we started looking at accompanying technology that aligns with our architectural strategy and goals. In this third part, we continue the solution from part II by focusing on what to do about adding a new service that may need to share data with the monolith (at least initially) and into some more complex deployment scenarios. We also explore consumer contract testing with Arquillian Algeron and how we can use that to get a handle on API changes in our services architecture.

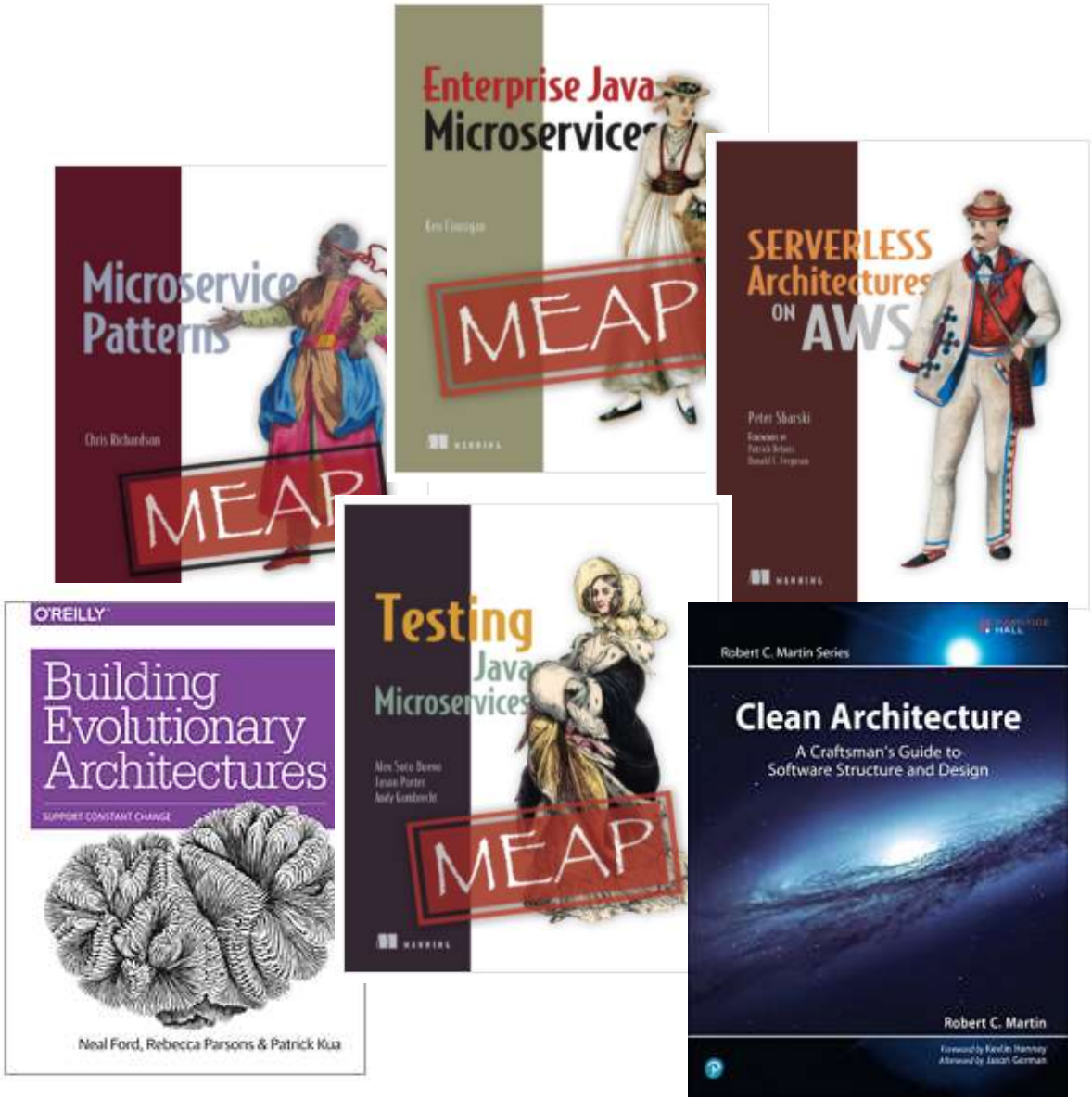
Follow along (@christianposta) on twitter or http://blog.christianposta.com for the latest updates and discussion.

You can find Part I and Part II.

Technologies

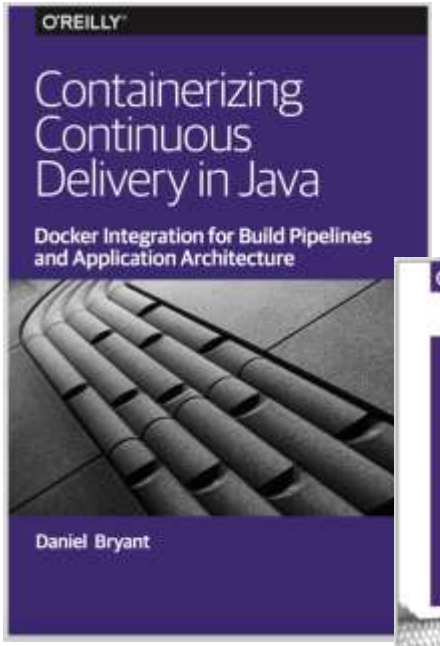
The technologies we'll use (in part II, III, and IV) to help guide us on this journey:

- Developer service frameworks (Spring Boot, WildFly, WildFly Swarm)
- API Design (APICur.io)
- Data frameworks (Spring Boot Teiid, Debezium.io)
- Integration tools (Apache Camel)
- Service mesh (Istio Service Mesh)
- Database migration tools (Liquibase)
- Dark launch / feature flag framework (FF4J)
- Deployment / CI-CD platform (Kubernetes / OpenShift)
- Kubernetes developer tools (Fabric8.io)
- Testing tools (Arquillian, Pact/Arquillian Algeron, Hoverfly, Spring Boot Test, RestAssured, Arquillian Cube)

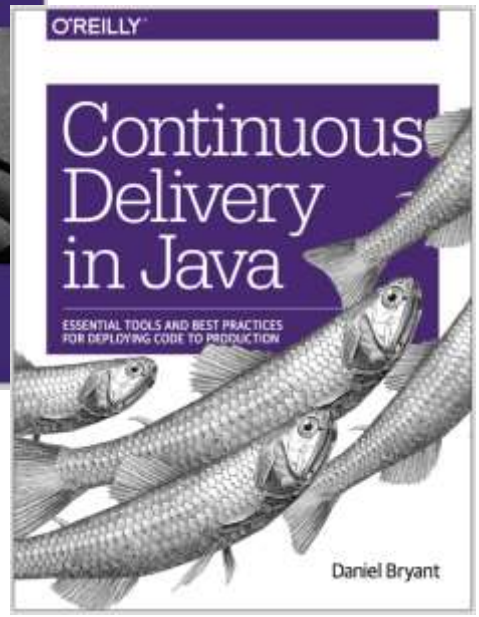


blog.christianposta.com/microservices/low-risk-monolith-to-microservice-evolution-part-iii/

Thanks for listening…



Coming soon!



bit.ly/2jWDSF7

Twitter: @danielbryantuk

Email: daniel.bryant@tai-dev.co.uk

Writing: https://www.infoq.com/profile/Daniel-Bryant

Talks: https://www.youtube.com/playlist?list=PLoVYf_0qOYNeBmrpjuBOOAqJnQb3QAEtM

10,00ft View: National/Regional

National and Regional Scale

- In Europe the rivers, mountains and coastline favoured balkanisation
- People banded together in tight groups, around culture and beliefs
- High turnover of states, due to trade, forced innovation, and conquest

National and Regional Scale

- This is the microservice approach to nation-building
  - Cohesive
  - Loosely-coupled
- Compare this to “monolithic” empires
  - Large geographical variation
  - Highly-coupled in terms of governance, economy, and social norms

Architecture fundamentals

- Coupling
  - ”Components have little or no knowledge of other components”
  - Interfaces (module interface, REST / RPC API)
  - Schema (RPC Payload, Message Schema, DB Schema)

- Cohesion
  - “Degree to which the elements within a component belong together”
  - Single responsibility principle (single reason to change)
  - Separation of concerns

	Monoliths	SOA	Microservices / SCS	FaaS / Serverless
Scope	Project	Enterprise	Product	Feature (or glue?)
Focus	Swiss Army Knife	Reuse, governance, control	Domain modelling, SRP, evolution	Function (in/out), rapid evolution
Organisation	Implemented and maintained (typically) by single team	Implemented by different org units. Maintenance done elsewhere	Services implemented and owned by product teams	Implemented by pioneers (hipsters?)
Deployment	Monolithic deployment	Monolithic orchestration of multiple services	Services deployed individually	Functions deployed individually
Management	None	Centralised	Distributed	Chaotic / Orchestrated
Inter-process communication	None	RPC or messaging, typically via middleware (ESB/MQ)	RPC via dumb pipes and smart endpoints, messaging/events	Events
Pioneers / stewards	Organisations, community or individuals	Enterprises and Vendors	Community and high perf organisations	Vendors/community
Core Architectural Constraints	Language and framework	Canonical domain model, standards	Interoperability	Cost (Gbs/ms)

	Monoliths	SOA	Microservices / SCS	FaaS / Serverless
Scope	Project	Enterprise	Product	Feature (or glue?)
Focus	Swiss Army Knife	Reuse, governance, control	Domain modelling, SRP, evolution	Function (in/out), rapid evolution
Organisation	Implemented and maintained (typically) by single team	Implemented by different org units. Maintenance done elsewhere	Services implemented and owned by product teams	Implemented by pioneers (hipsters?)
Deployment	Monolithic deployment	Monolithic orchestration of multiple services	Services deployed individually	Functions deployed individually
Management	None	Centralised	Distributed	Chaotic / Orchestrated
Inter-process communication	None	RPC or messaging, typically via middleware (ESB/MQ)	RPC via dumb pipes and smart endpoints, messaging/events	Events
Pioneers / stewards	Organisations, community or individuals	Enterprises and Vendors	Community and high perf organisations	Vendors/community
Core Architectural Constraints	Language and framework	Canonical domain model, standards	Interoperability	Cost (Gbs/ms)

Coupling, Cohesion and Continuous Delivery

- Design
  - Cohesion makes reasoning easy
  - Loose coupling reduces impact
- Build, unit and integration
  - Cohesion limits dependencies
  - Loose coupling allows simulation
- End-to-end tests
  - Cohesion/coupling orchestration
- Deployment
  - Cohesion minimises number of components in play
  - Coupling leads to “monoliths”
- Observability
  - Cohesive is easier to understand
  - High coupling typically leads to large blast radius and “murder mystery” style debugging

Coupling, Cohesion and Microservices

Programming Techniques — B. Morris, Editor

On the Criteria To Be Used in Decomposing Systems into Modules

D.L. Parnas
Carnegie-Mellon University

www.cs.umd.edu/class/spring2003/cmsc838p/Design/criteria.pdf

the morning paper

On the criteria to be used in decomposing systems into modules

https://blog.acolyer.org/2016/09/05/on-the-criteria-to-be-used-in-decomposing-systems-into-modules/

National and Regional Scale

- With microservices, coupling and cohesion is “turtles all the way down”
- Avoid the monolith where appropriate
  - Application
  - Deployment
  - Databases/middleware
  - Thinking