MODERN ENTERPRISE ARCHITECTURE: ARCHITECTING FOR OUTCOMES

Simon Rohrer

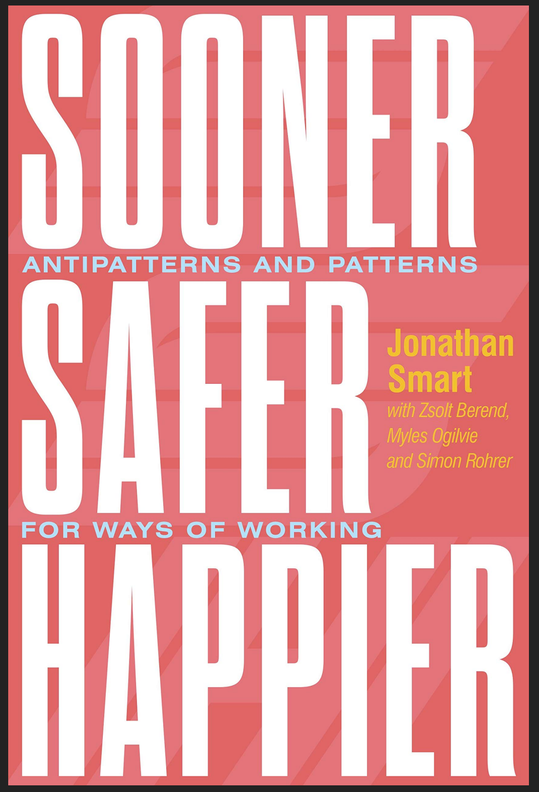
MY JOURNEY

DEVELOPER

ENTERPRISE ARCHITECT

WAYS OF WORKING LEAD

HEAD OF ENT. ARCH & WOW



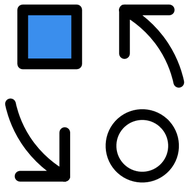
MODERN ENTERPRISE



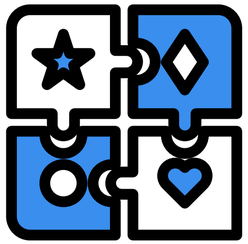
Teams-of-Teams-of-Teams: > 150 Employees



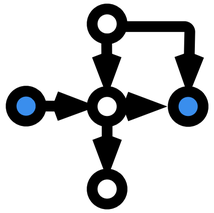
Interact with Your Customers Digitally



Change is a Constant

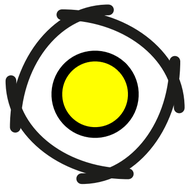
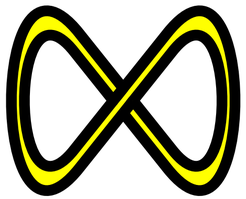
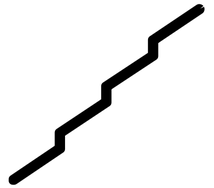


Heterogeneous Environment: Old/New, Big/Small, Slow/Fast, Monolith/Distributed Vendors/Internal

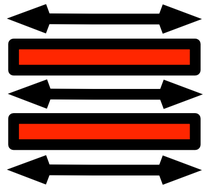
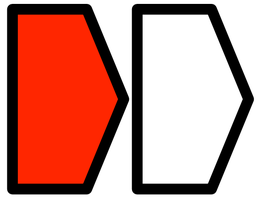
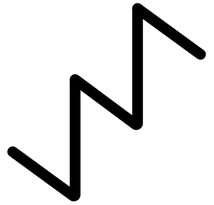


Systems of Systems of Systems

Modern Enterprise Architecture

A	B	C	D	E
				
Aligning Value, People and Technology	Better Value Sooner Safer Happier	Continuous Conversational Governance	DevOps at Enterprise Scale	Evolutionary Enterprise Architecture

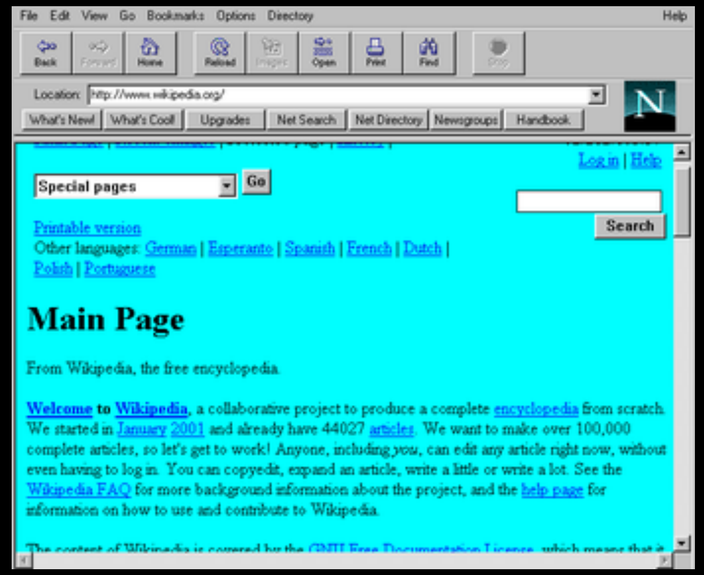
rather than…

				
Bilateral Alignments	De-duplication Standardisation Cost reduction	Phase-gated Document-led Adversarial Governance	Change not Run Dev not Ops Project not Product	Entropy + Big Bang Improvements

WHERE ARE WE & HOW DID WE GET HERE?

1990S

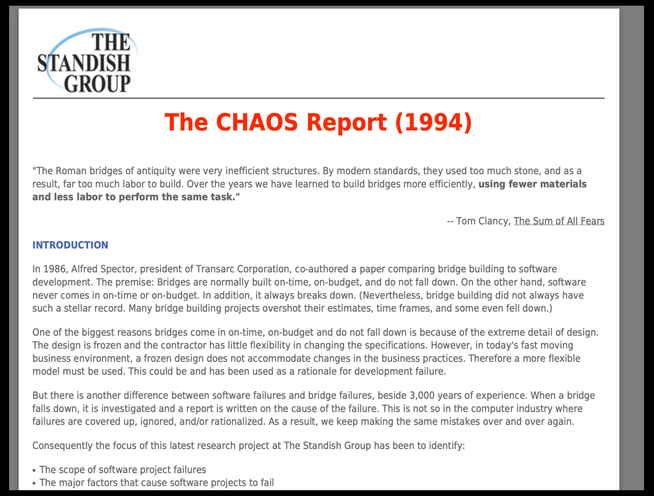
1994



1999



1994



THE STANDISH GROUP

The CHAOS Report (1994)

"The Roman bridges of antiquity were very inefficient structures. By modern standards, they used too much stone, and as a result, far too much labor to build. Over the years we have learned to build bridges more efficiently, **using fewer materials and less labor to perform the same task.**"

-- Tom Clancy, The Sum of All Fears

INTRODUCTION

In 1986, Alfred Spector, president of Transarc Corporation, co-authored a paper comparing bridge building to software development. The premise: Bridges are normally built on-time, on-budget, and do not fall down. On the other hand, software never comes in on-time or on-budget. In addition, it always breaks down. (Nevertheless, bridge building did not always have such a stellar record. Many bridge building projects overshot their estimates, time frames, and some even fell down.)

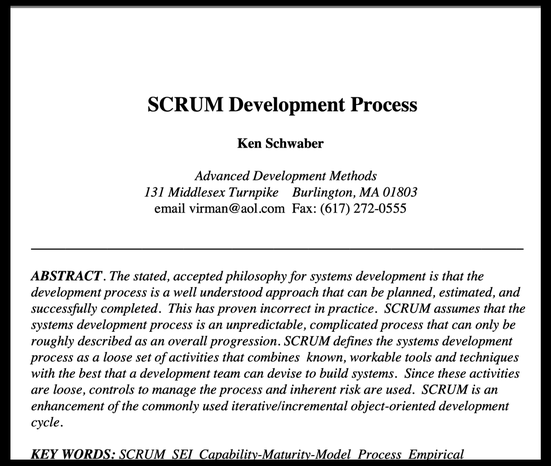
One of the biggest reasons bridges come in on-time, on-budget and do not fall down is because of the extreme detail of design. The design is frozen and the contractor has little flexibility in changing the specifications. However, in today's fast moving business environment, a frozen design does not accommodate changes in the business practices. Therefore a more flexible model must be used. This could be and has been used as a rationale for development failure.

But there is another difference between software failures and bridge failures, beside 3,000 years of experience. When a bridge falls down, it is investigated and a report is written on the cause of the failure. This is not so in the computer industry where failures are covered up, ignored, and/or rationalized. As a result, we keep making the same mistakes over and over again.

Consequently the focus of this latest research project at The Standish Group has been to identify:

- The scope of software project failures
- The major factors that cause software projects to fail

1995



SCRUM Development Process

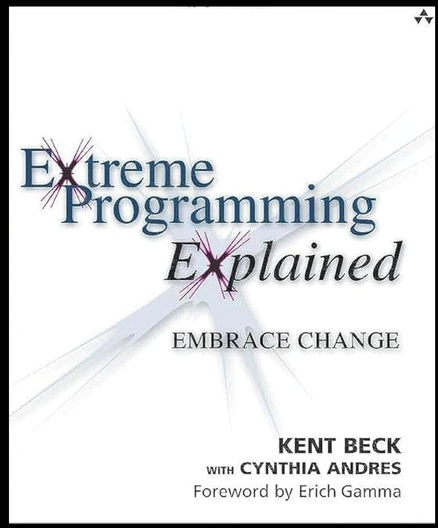
Ken Schwaber

Advanced Development Methods
131 Middlesex Turnpike Burlington, MA 01803
email virman@aol.com Fax: (617) 272-0555

***ABSTRACT**. The stated, accepted philosophy for systems development is that the development process is a well understood approach that can be planned, estimated, and successfully completed. This has proven incorrect in practice. SCRUM assumes that the systems development process is an unpredictable, complicated process that can only be roughly described as an overall progression. SCRUM defines the systems development process as a loose set of activities that combines known, workable tools and techniques with the best that a development team can devise to build systems. Since these activities are loose, controls to manage the process and inherent risk are used. SCRUM is an enhancement of the commonly used iterative/incremental object-oriented development cycle.*

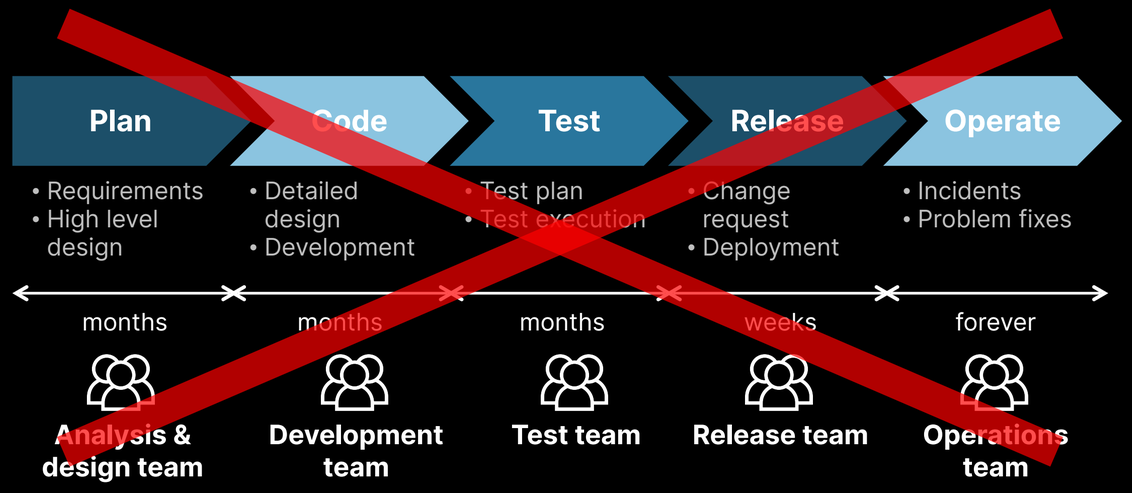
***KEY WORDS:** SCRUM, SEI Capability-Maturity-Model, Process, Empirical*

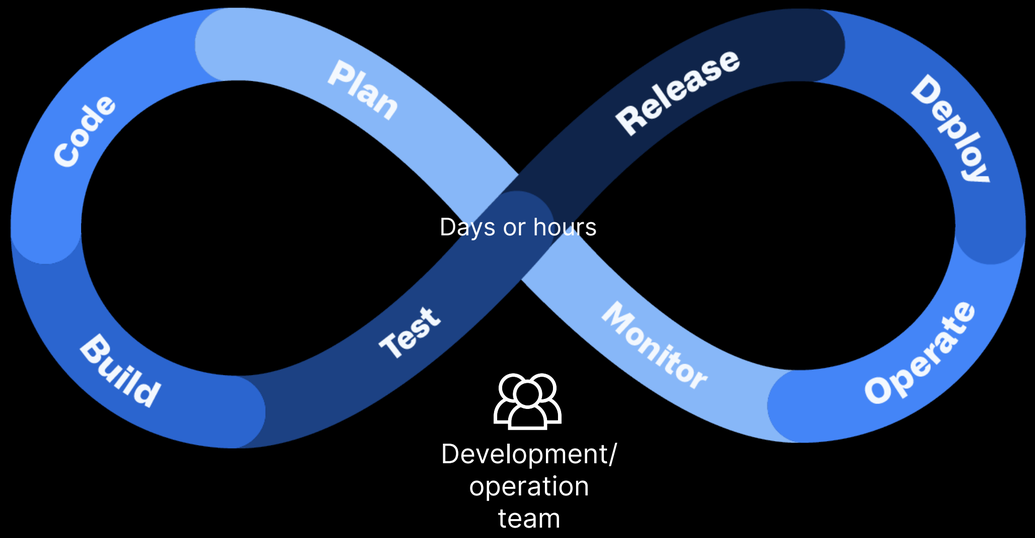
1999



SOFTWARE ATE THE WORLD

A PARADIGM SHIFT HAPPENED IN PRODUCT DEVELOPMENT



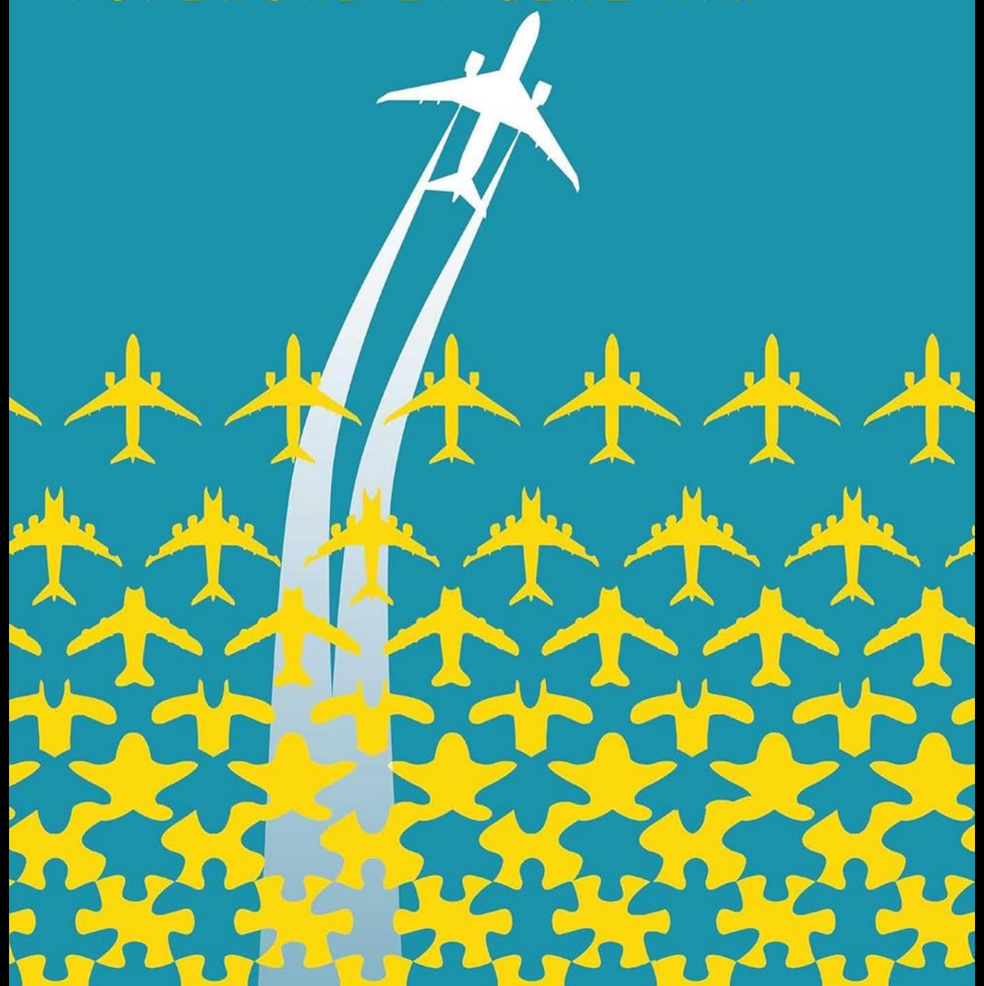


PROJECT TO PRODUCT

MIK KERSTEN

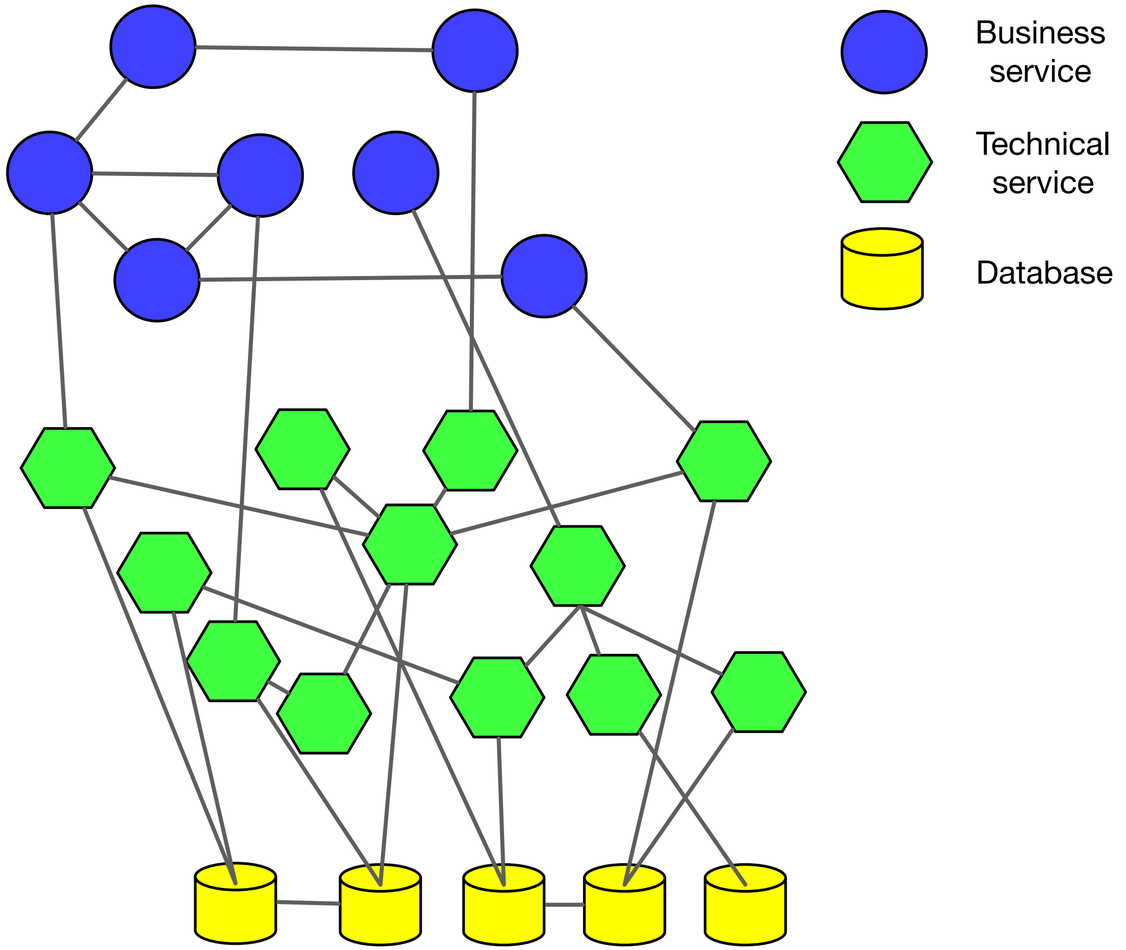
FOREWORD BY GENE KIM

HOW TO SURVIVE AND THRIVE IN THE AGE OF DIGITAL DISRUPTION WITH THE FLOW FRAMEWORK

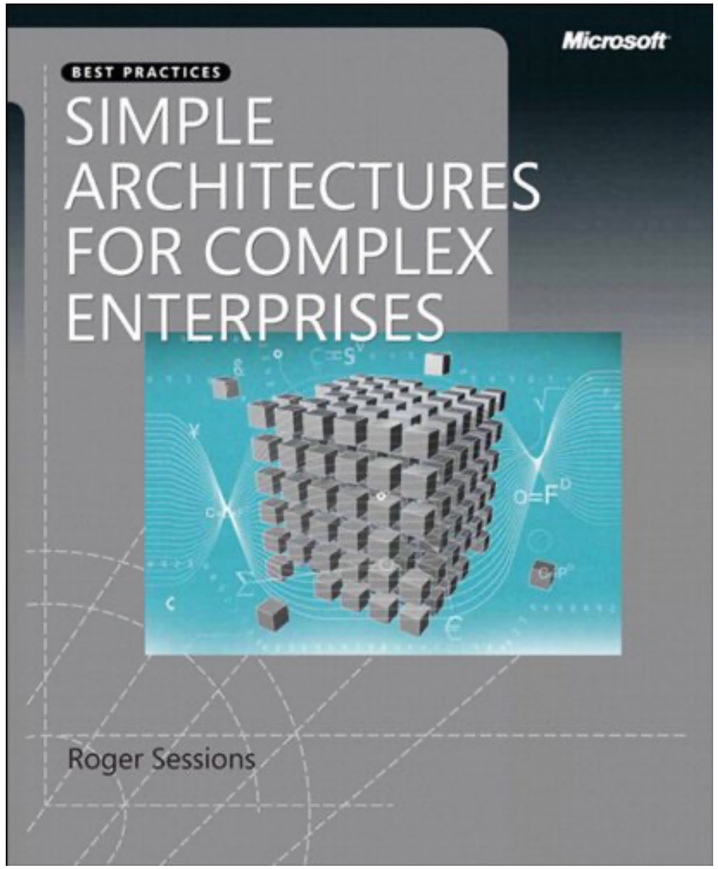


NOT EVERYONE NOTICED

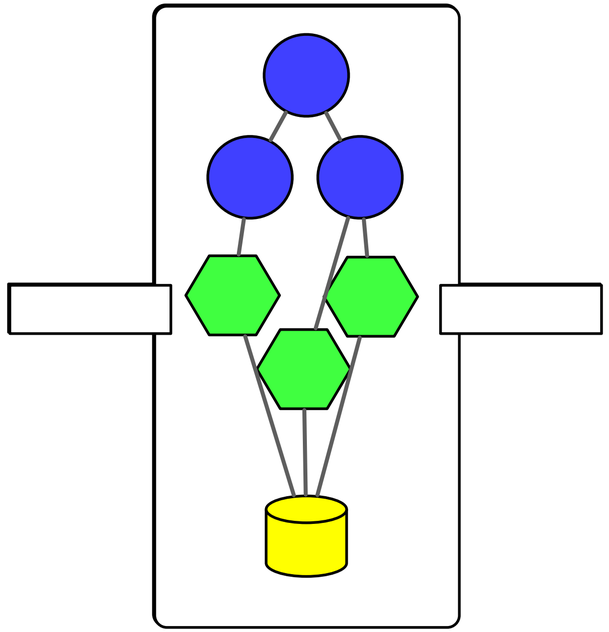
COMPLEXITY IN THE EXISTING (“LEGACY”) SYSTEM LANDSCAPE STARTED TO DOMINATE THE PACE OF CHANGE



PATTERNS WERE DOCUMENTED TO MANAGE COMPLEXITY BUT NOT EVERYONE USED THEM. THEY STILL DON'T.



2008



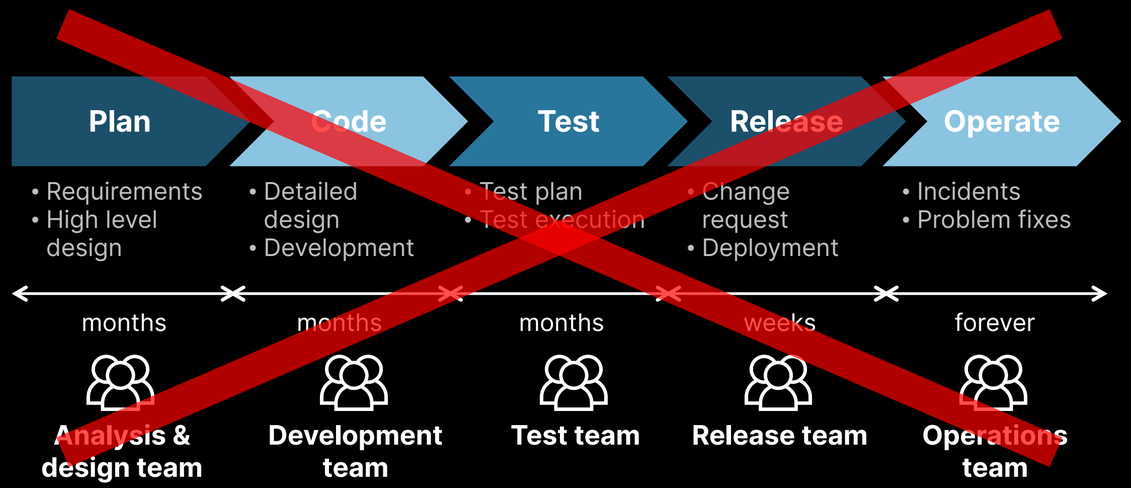
Autonomous business capability

1. Autonomy
2. Explicit boundaries
3. Partitioning of functionality
4. Dependencies defined by functionality
5. Asynchronicity [by default]
6. Partitioning of data
7. No cross-fortress transactions
8. Single point security
9. Inside trust
10. **Keep it simple**

OUR PROBLEMS ARE SIMILAR BUT DIFFERENT TO THOSE IN THE 1990S

1990s

Civil engineering processes are the best way to deliver any change



2020s

The enterprise landscape is so complex we need to use project management techniques to coordinate multiple teams for simple changes

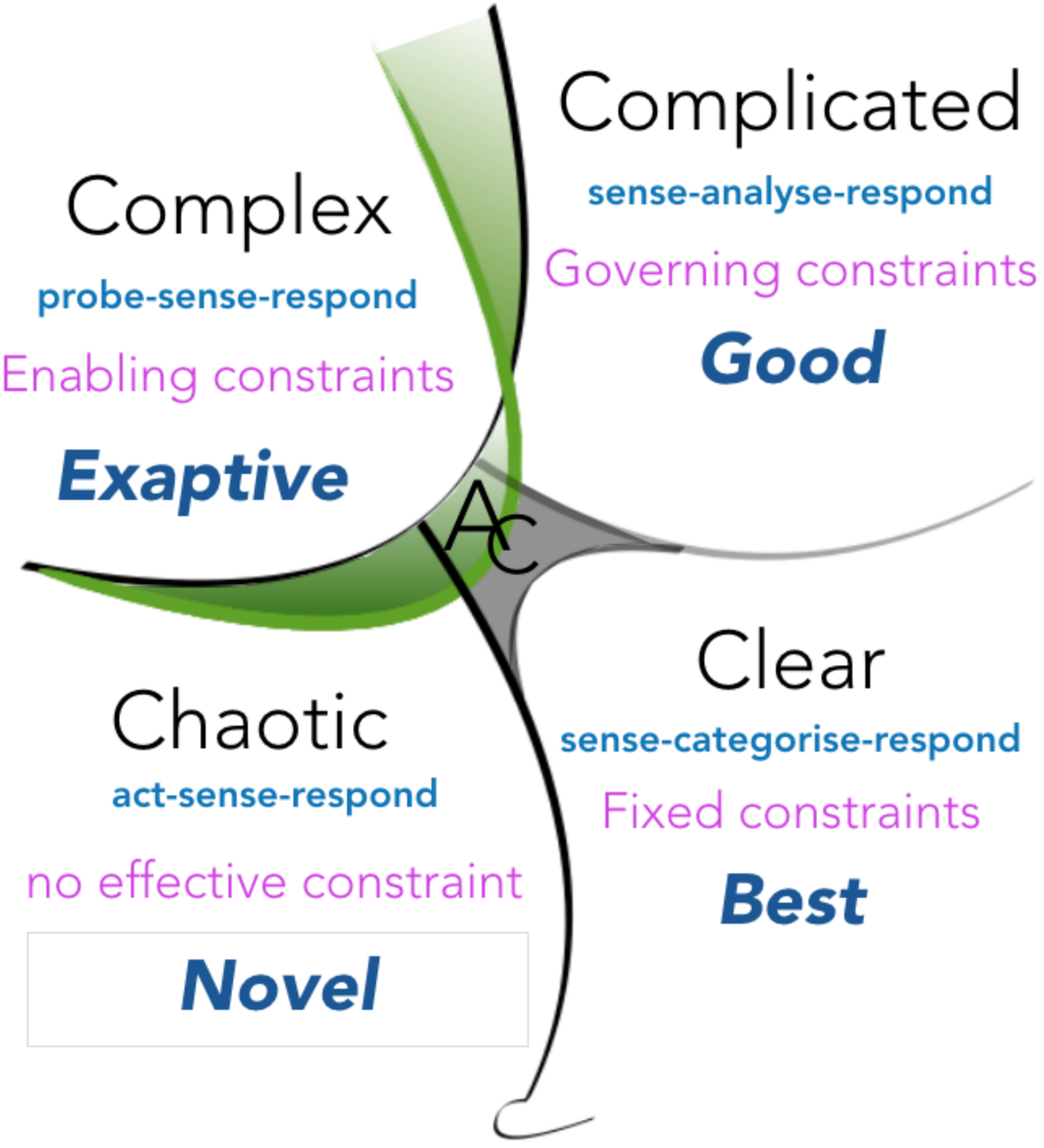
“AN ORGANIZATION THAT LACKS A VIABLE PROGRAM IN ENTERPRISE ARCHITECTURE WILL PAY A SEVERE COST IN I.T. COMPLEXITY.

COMPLEXITY IS THE ENEMY.

ENTERPRISE ARCHITECTURE IS THE SOLUTION.

THE ONLY SOLUTION.”

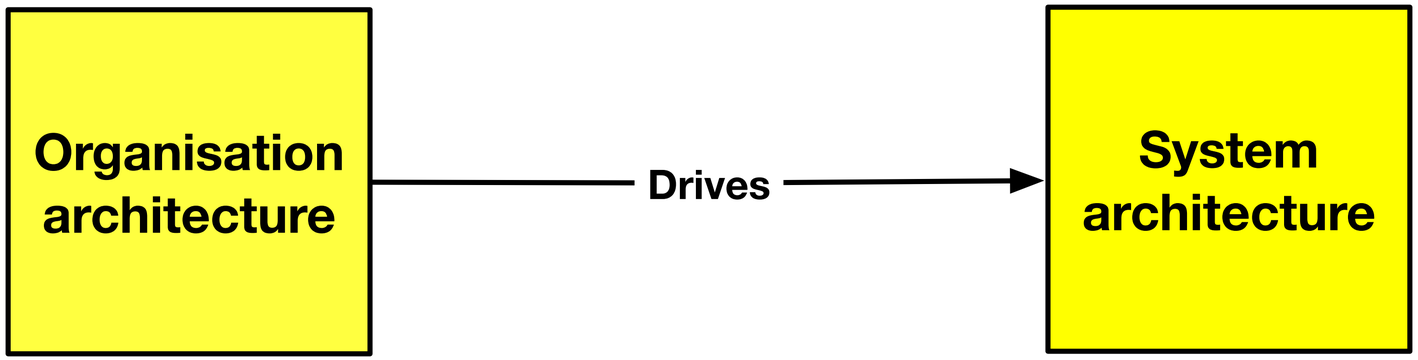
Roger Sessions, The Job of the Enterprise Architect (2008)


Complicated
sense-analyse-respond
Governing constraints
Good
Complex
probe-sense-respond
Enabling constraints
Exaptive
AC
Clear
sense-categorise-respond
Fixed constraints
Best
Chaotic
act-sense-respond
no effective constraint
Novel

A

ALIGNING VALUE, PEOPLE & TECHNOLOGY

CONWAY'S LAW (VERY ROUGHLY)



“IF THE ARCHITECTURE OF THE SYSTEM AND THE ARCHITECTURE OF THE ORGANIZATION ARE AT ODDS, THE ARCHITECTURE OF THE ORGANIZATION WINS.”

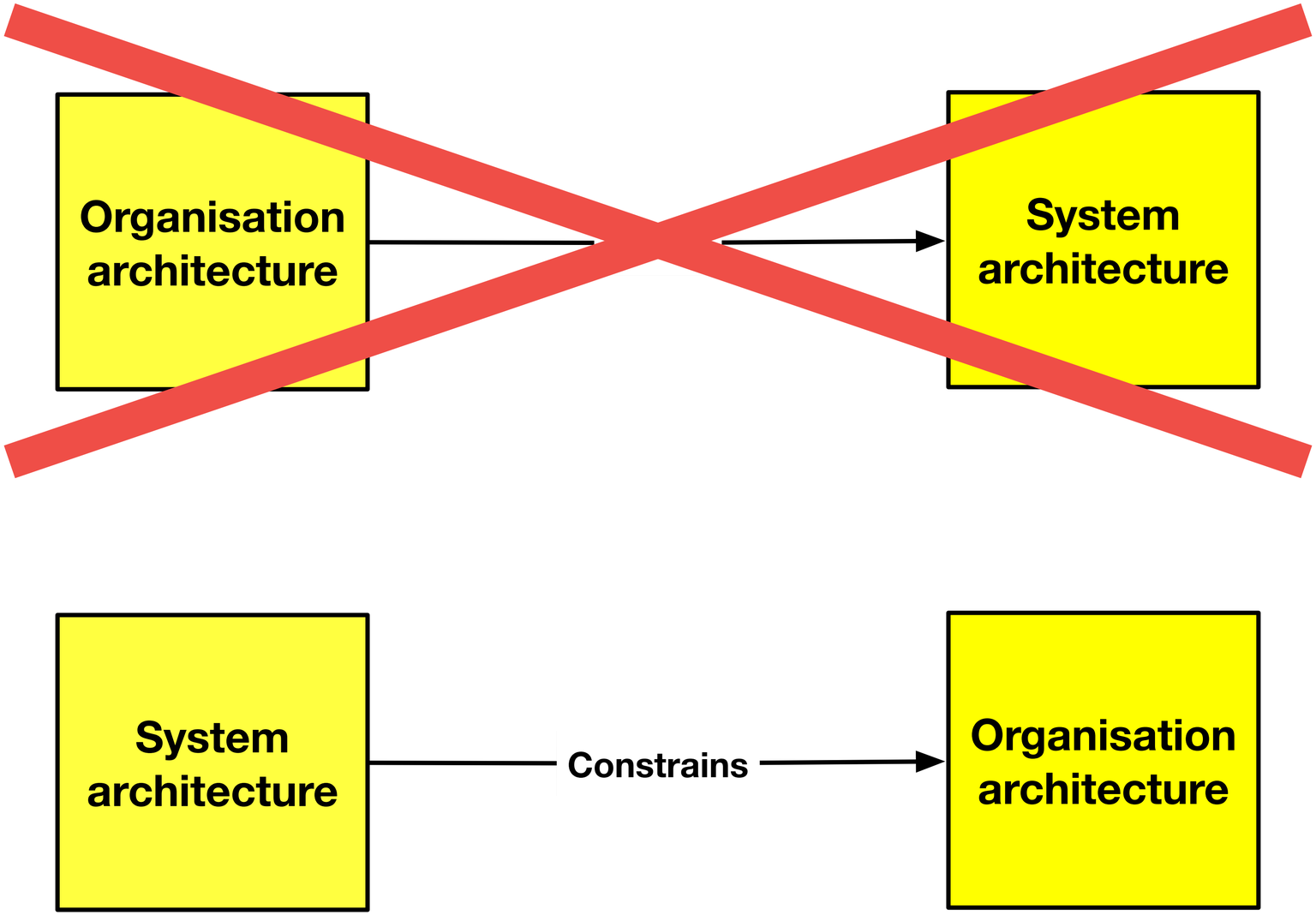
Ruth Malan, Conway’s Law (2008)

BUT!

“WHERE ALL HELL REALLY BREAKS LOOSE IS WHEN MANAGEMENT DECIDES TO REORGANISE THINGS.

… IF YOU TRY TO CHANGE THE ORGANISATION, THE SOFTWARE WON’T LET IT HAPPEN.”

Allan Kelly, Continuous Delivery & Conway’s Law (2018)


Organisation architecture
System architecture
System architecture
Constrains
Organisation architecture

???

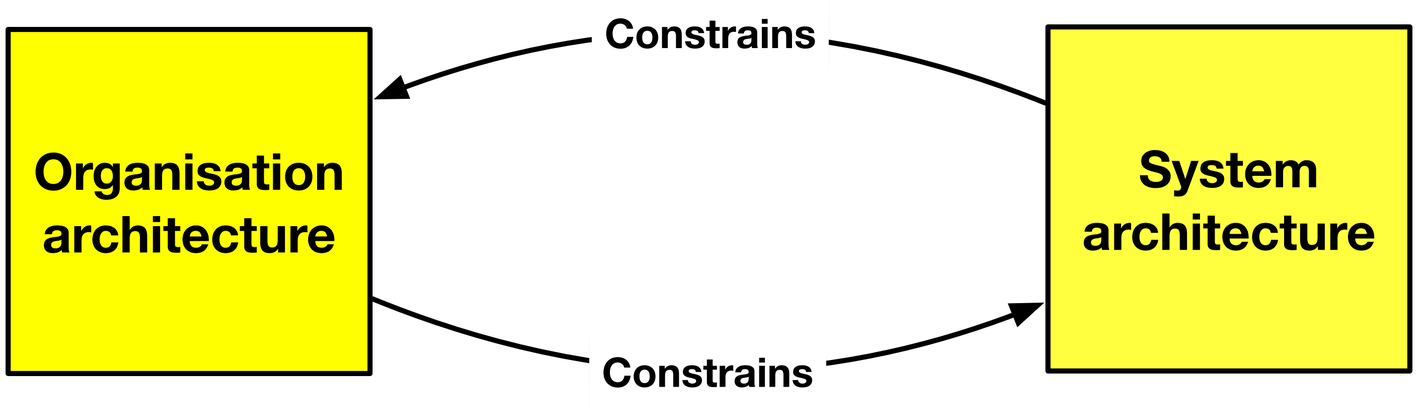


“WE SHAPE OUR BUILDINGS AND AFTERWARD OUR BUILDINGS SHAPE US”

Winston Churchill, speech to House of Lords (1943)

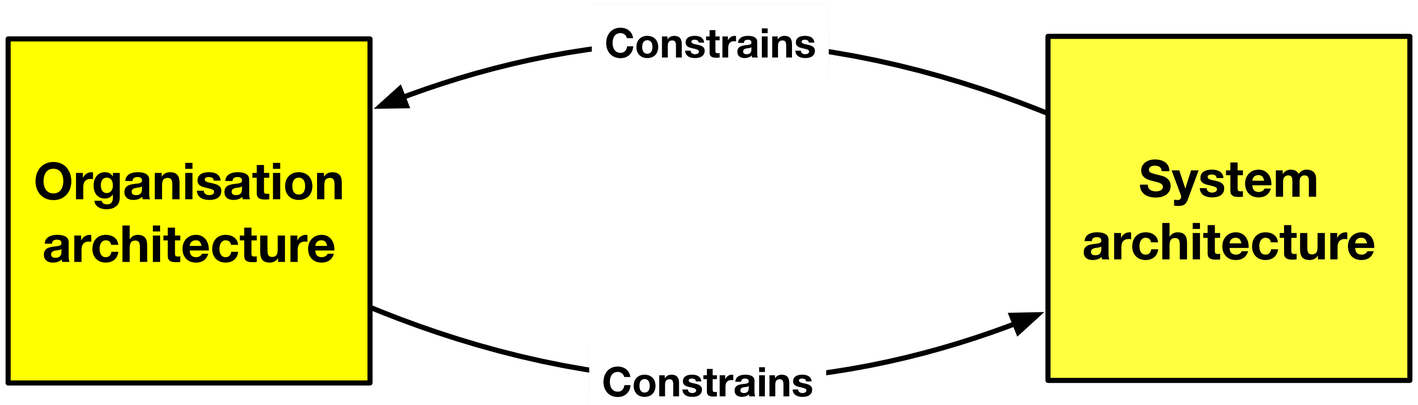
“WE SHAPE OUR ARCHITECTURE AND THEN THE ARCHITECTURE SHAPES US”

Gene Kim, DevOps Masterclass (2023)


Organisation architecture
Constrains
Constrains
System architecture

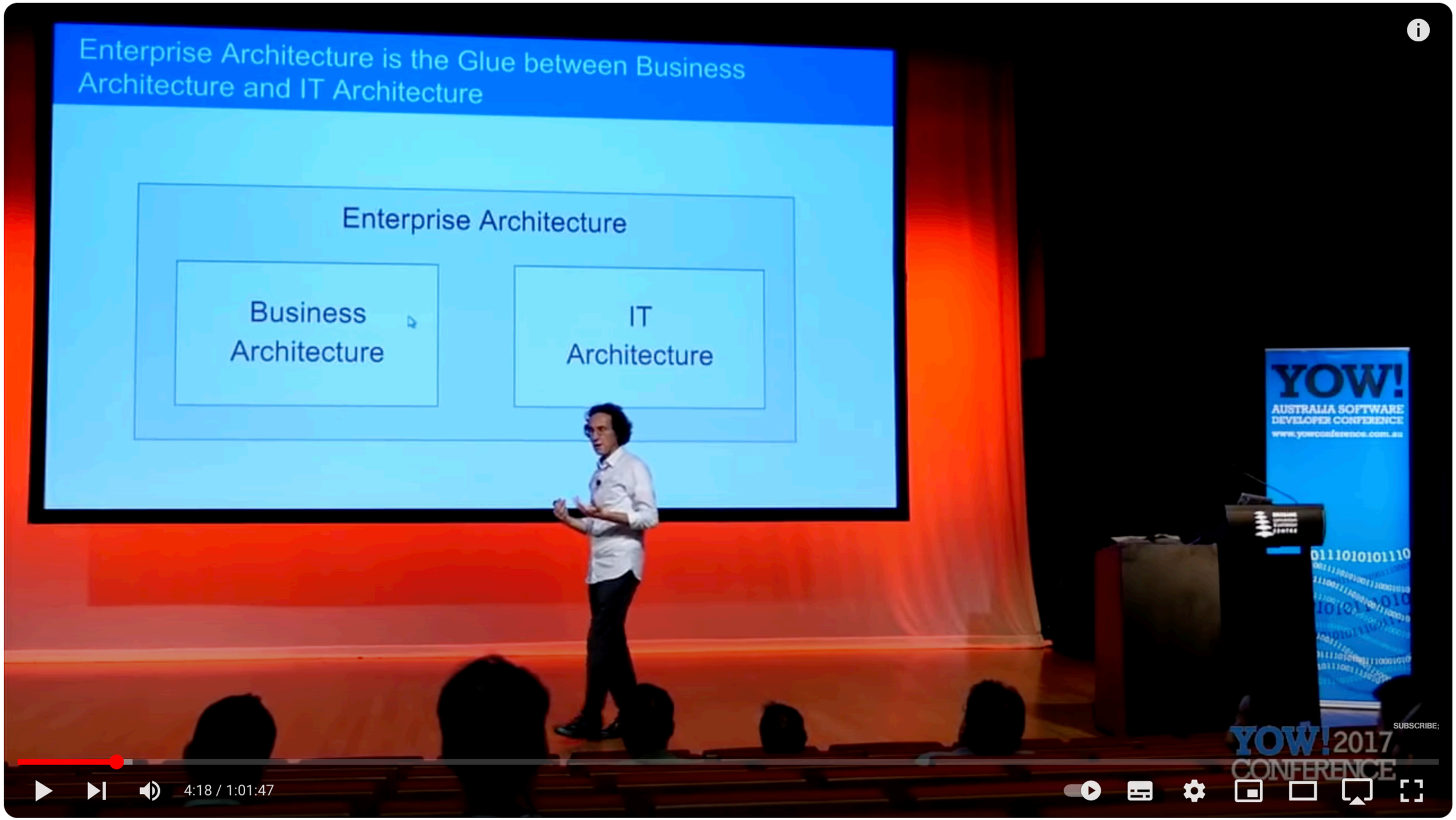
“SYSTEM ARCHITECTS (WHO WE CALL ARCHITECTS) AND BUSINESS/ORGANIZATION ARCHITECTS (WHO WE CALL MANAGERS) SHOULD NOT WORK AS IF ONE HAS NO IMPACT ON THE OTHER.”

Ruth Malan, Conway’s Law (2008)

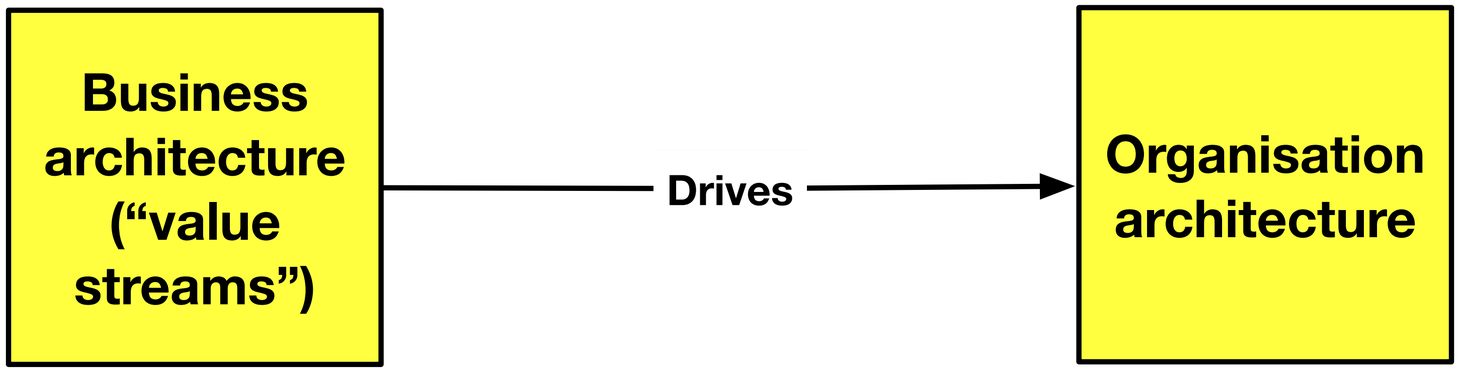


Organization architects
(“managers”)

System architects
(“architects”)



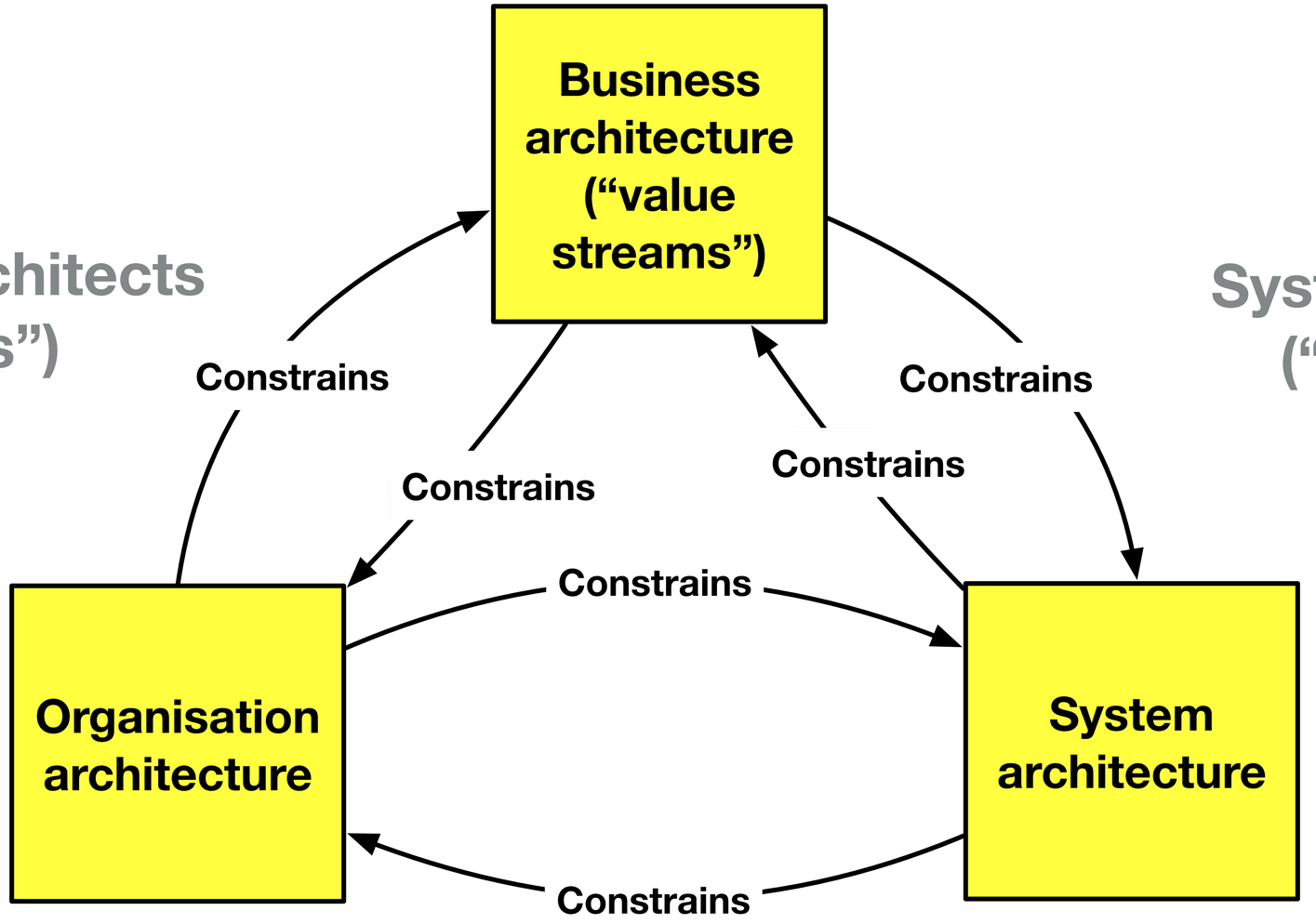
Enterprise Architecture = Architecting the Enterprise? • Gregor Hohpe • YOW! 2017



Organization architects
(“managers”)

Organization architects
(“managers”)

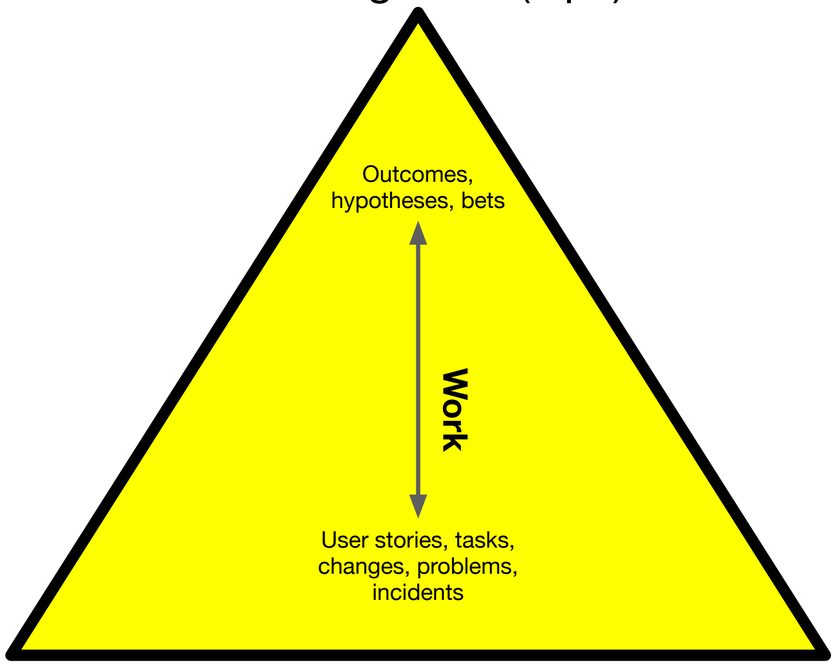
System architects
(“architects”)



ALIGNING ALL 3 FOR SUSTAINABLE FLOW

Value

OKRs looking forward (Dev)
KPIs & OLIs right now &
looking back (Ops)



Organization

People, teams,
teams-of-teams,
departments

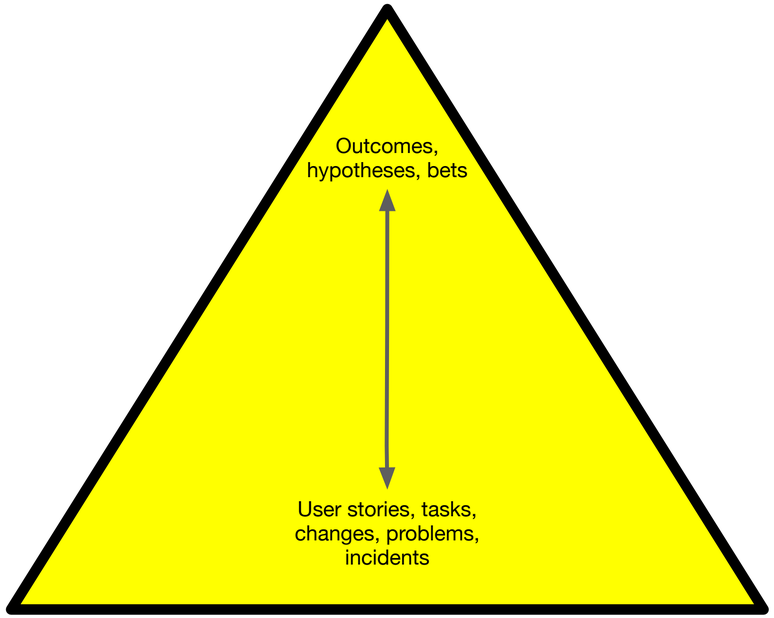
Technology & processes

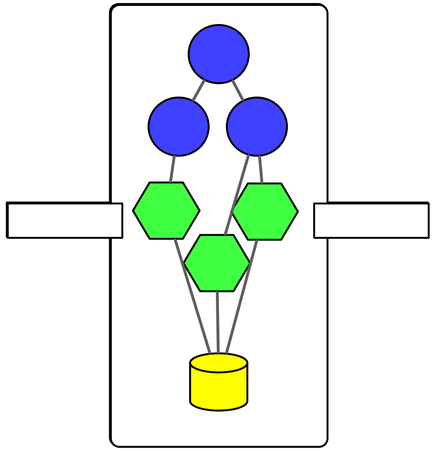
Business services &
business processes,
IT services & IT
components

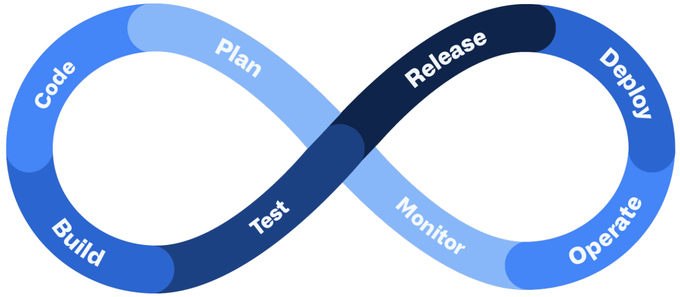
Level 0: an ideal 2-pizza stream-aligned team

Value

OKRs looking forward (Dev)
KPIs & OLIs right now &
looking back (Ops)





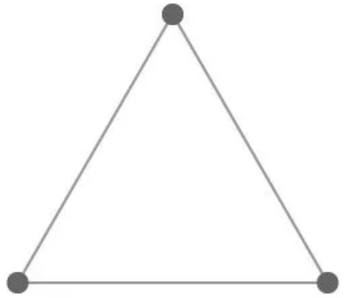


2 pizza team - <?20 people

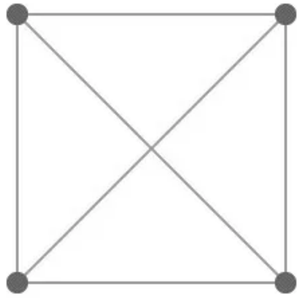
Full stack, full lifecycle, full burrito, T-shaped people - YBIYRI
Self-organizing

Software that fits inside the team's head

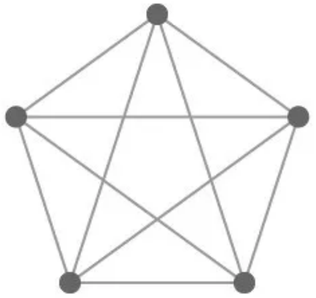
DDD
Eventually consistent
Independently deployable & testable
"Monolith vs Microservice is missing the point"
Emergent design & evolutionary* architecture



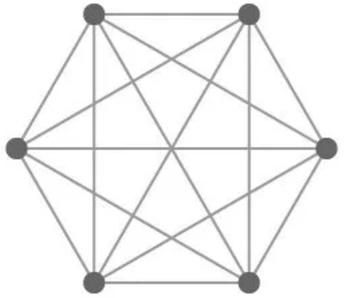
3 people, 3 lines



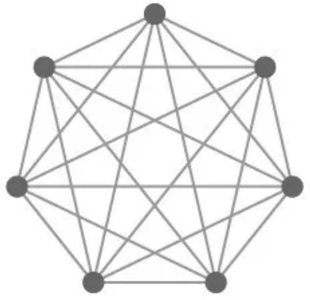
4 people, 6 lines



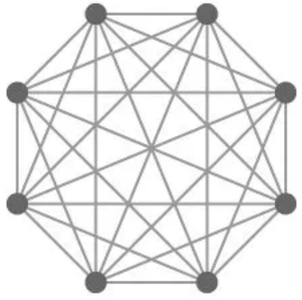
5 people, 10 lines



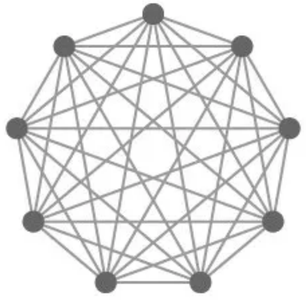
6 people, 15 lines



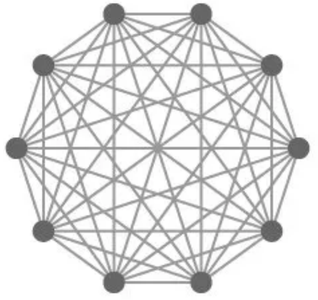
7 people, 21 lines



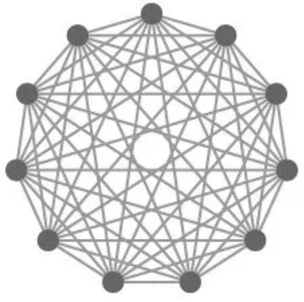
8 people, 28 lines



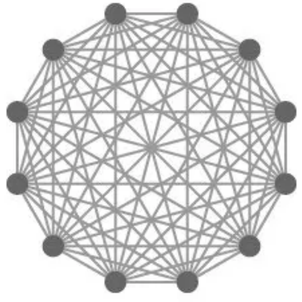
9 people, 36 lines



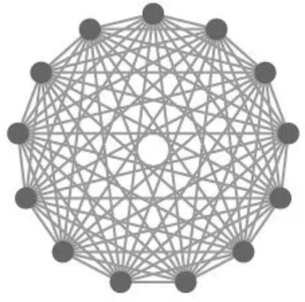
10 people, 45 lines



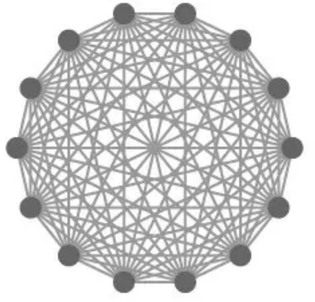
11 people, 55 lines



12 people, 66 lines



13 people, 78 lines



14 people, 91 lines

design

Editor: Martin Fowler ■ ThoughtWorks ■ fowler@acm.org

Who Needs an Architect?

Martin Fowler



Wandering down our corridor a while ago, I saw my colleague Dave Rice in a particularly grumpy mood. My brief question caused a violent statement, "We shouldn't interview anyone who has 'architect' on his resume." At first blush, this was an odd turn of phrase, because we usually introduce Dave as one of our leading architects.

The reason for his title schizophrenia is the fact that, even by our industry's standards, "architect" and "architecture" are terribly overloaded words. For many, the term "software architect" fits perfectly with the smug controlling image at the end of *Matrix Reloaded*. Yet even in firms that have the greatest contempt for that image, there's a vital role for the technical leadership that an architect such as Dave plays.

What is architecture?

When I was fretting over the title for *Patterns of Enterprise Application Architecture* (Addison-Wesley, 2002), everyone who reviewed it agreed that "architecture" belonged in the title. Yet we all felt uncomfortable defining the word. Because it was my book, I felt compelled to take a stab at defining it.

chitect.) However, as so often occurs, inside the blighted cynicism is a pinch of truth. Understanding came to me after reading a posting from Ralph Johnson on the Extreme Programming mailing list. It's so good I'll quote it all.

A previous posting said

> The RUP, working off the IEEE definition, defines architecture as "the highest level concept of a system in its environment. The architecture of a software system (at a given point in time) is its organization or structure of significant components interacting through interfaces, those components being composed of successively smaller components and interfaces."

Johnson responded:

> I was a reviewer on the IEEE standard that used that, and I argued uselessly that this was clearly a completely bogus definition. There is no highest level concept of a system. Customers have a different concept than developers. Customers do not care at all about the structure of significant components. So, perhaps an architecture is the highest level concept that developers have of a system in its environment. Let's forget the developers who just understand their little piece. Architecture is the highest level concept of the expert developers. What makes a component

Is Design Dead?

For many that come briefly into contact with Extreme Programming, it seems that XP calls for the death of software design. Not just is much design activity ridiculed as "Big Up Front Design", but such design techniques as the UML, flexible frameworks, and even patterns are de-emphasized or downright ignored. In fact XP involves a lot of design, but does it in a different way than established software processes. XP has rejuvenated the notion of evolutionary design with practices that allow evolution to become a viable design strategy. It also provides new challenges and skills as designers need to learn how to do a simple design, how to use refactoring to keep a design clean, and how to use patterns in an evolutionary style.

May 2004



Martin Fowler

POPULAR

DESIGN

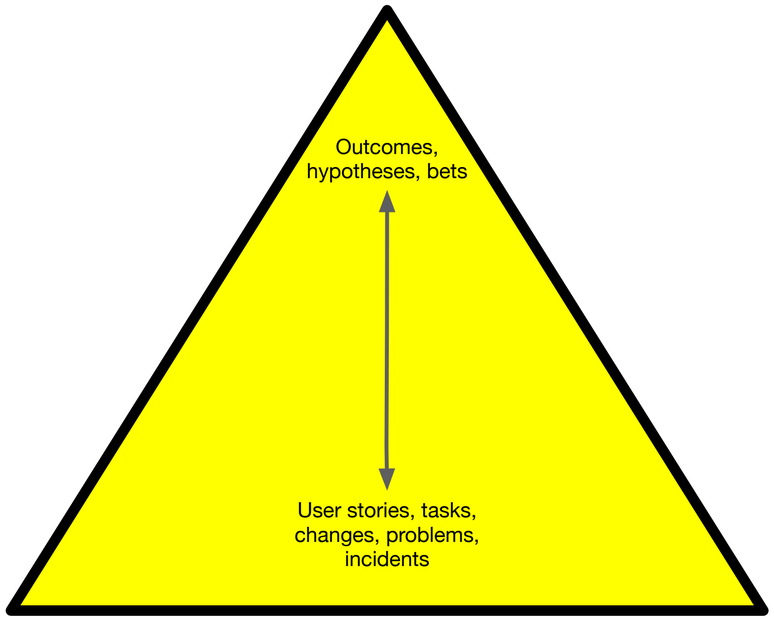
AGILE

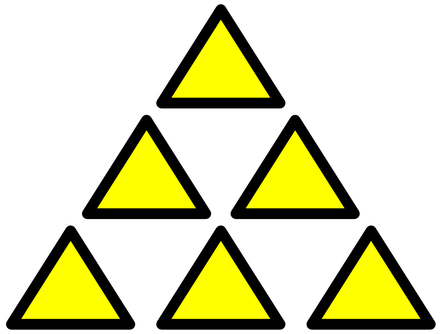
EXTREME PROGRAMMING

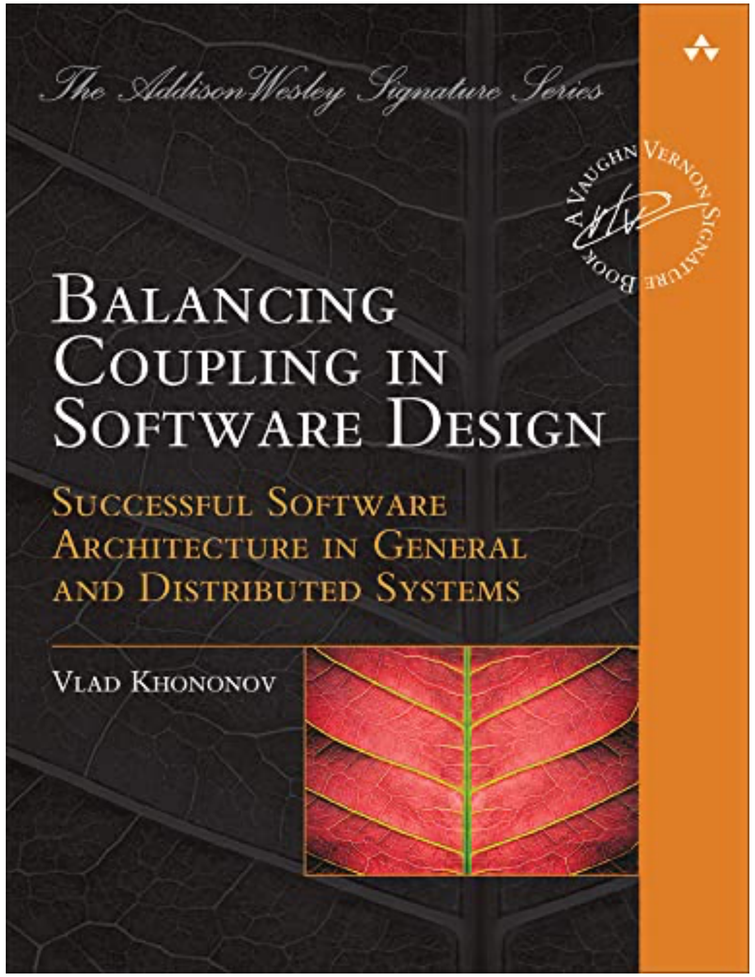
EVOLUTIONARY DESIGN

CONTENTS

Planned and Evolutionary Design
The Enabling Practices of XP
The Value of Simplicity
What on Earth is Simplicity Anyway
Does Refactoring Violate YAGNI?
Patterns and XP
Growing an Architecture
UML and XP
On Metaphor
Do you wanna be an Architect when you grow up?
Reversibility
The Will to Design
Things that are difficult to refactor in
Is Design Happening?
So is Design Dead?


Outcomes,
hypotheses, bets
User stories, tasks,
changes, problems,
incidents

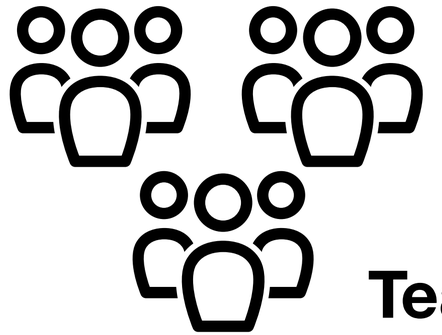


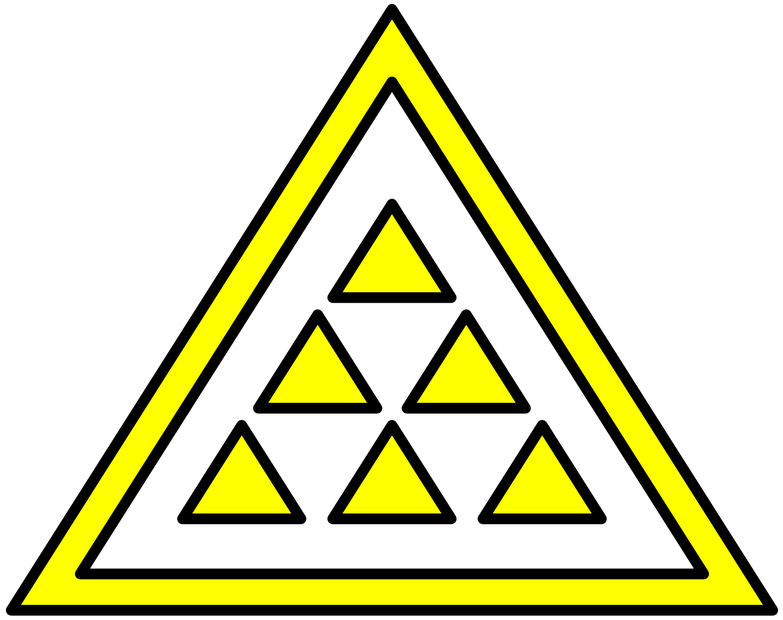

The Addison Wesley Signature Series
A VAUGHN VERNON SIGNATURE BOOK
BALANCING COUPLING IN SOFTWARE DESIGN
SUCCESSFUL SOFTWARE ARCHITECTURE IN GENERAL AND DISTRIBUTED SYSTEMS
VLAD KHONONOV

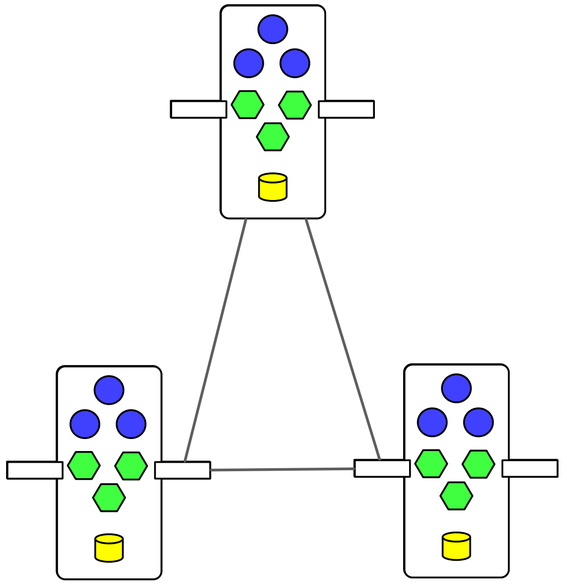
TEAM-OF-TEAMS

Value

OKRs looking forward (Dev)
KPIs & OLIs right now &
looking back (Ops)







Team-of-teams?
<150? 200? ???

"Tribe"
Nested value
Self-organizing?

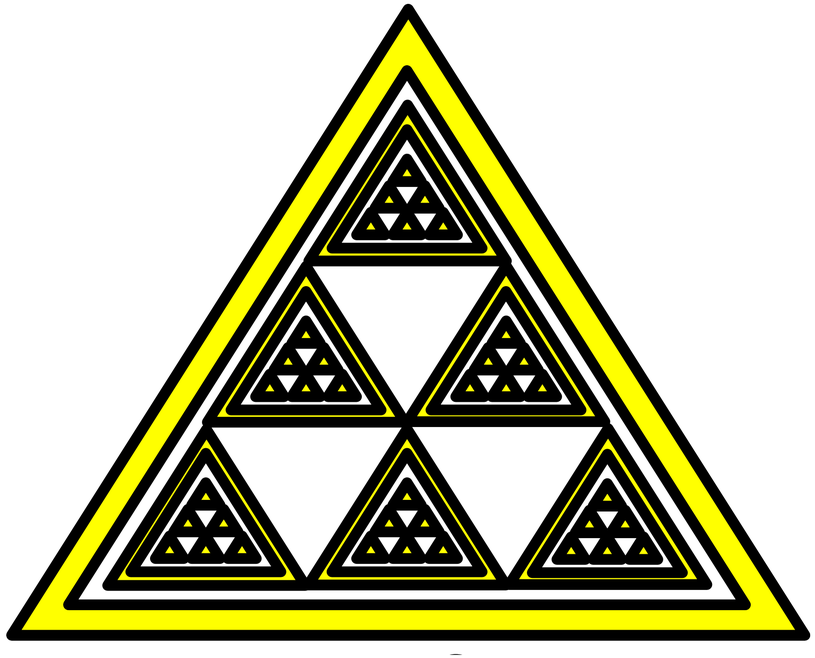
System-of-systems & domain that fits inside the team's head

Nested DDD
Shared data model?
Eventually consistent
Each component independently deployable & testable

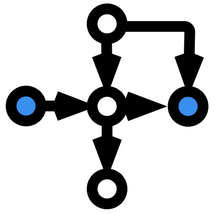
A LINE OF BUSINESS

Value

OKRs looking forward (Dev)
KPIs & OLIs right now &
looking back (Ops)







Team-of-team-of-teams
1000–2000??

Department /
business line
Nested value
Self-organizing???

System-of-system-of-systems
Does this fit in anyone's head?

Further nested DDD / Business capability (group)?
Minimally shared data model - just keys?
Eventually consistent between subdomains
Key paths and failure modes should be known

“AS THE COMPLEXITY OF A SYSTEM INCREASES, THE ACCURACY OF ANY SINGLE AGENT'S OWN MODEL OF THAT SYSTEM DECREASES RAPIDLY.”

David Woods, Report from the SNAFUcatchers Workshop on Coping With Complexity (2017)

conclusion

The basic thesis of this article is that organizations which design systems (in the broad sense used here) are constrained to produce designs which are copies of the communication structures of these organizations. We have seen that this fact has important implications for the management of system design. Primarily, we have found a criterion for the structuring of design organizations: a design effort should be organized according to the need for communication.

This criterion creates problems because the need to communicate at any time depends on the system concept in effect at that time. Because the design which occurs first is almost never the best possible, the prevailing system concept may need to change. Therefore, flexibility of organization is important to effective design.

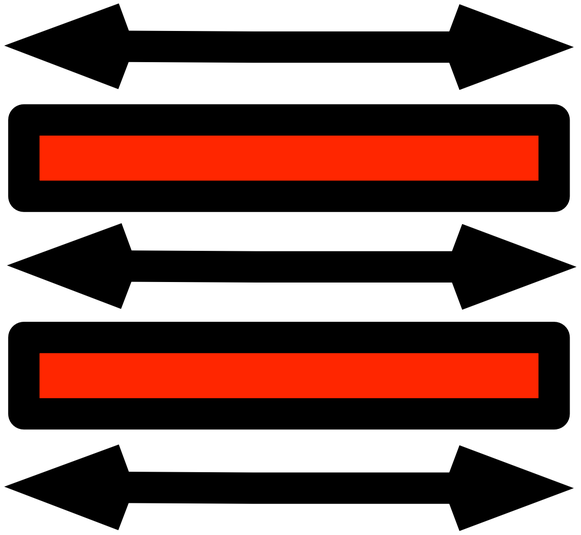
Ways must be found to reward design managers for keeping their organizations lean and flexible. There is need for a philosophy of system design management which is not based on the assumption that adding manpower simply adds to productivity. The development of such a philosophy promises to unearth basic questions about value of resources and techniques of communication which will need to be answered before our system-building technology can proceed with confidence. ■



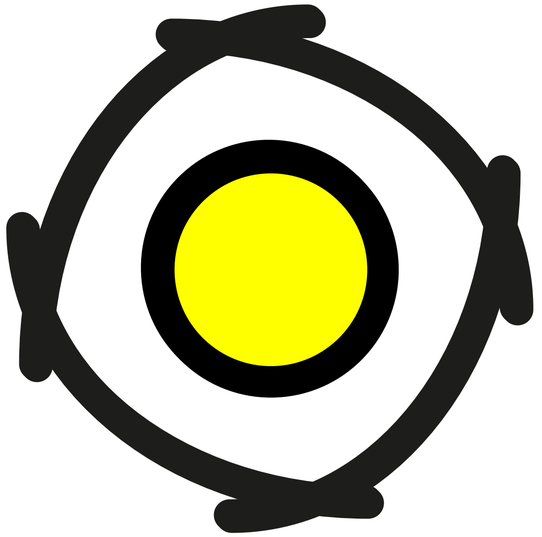
ALIGNING
VALUE, PEOPLE
& TECHNOLOGY

A

RATHER THAN



BILATERAL
ALIGNMENT



B

BETTER VALUE SOONER
SAFER HAPPIER

ARCHITECTING FOR OUTCOMES

ANTIPATTERN: TRADITIONAL EA OUTCOMES

Standardisation

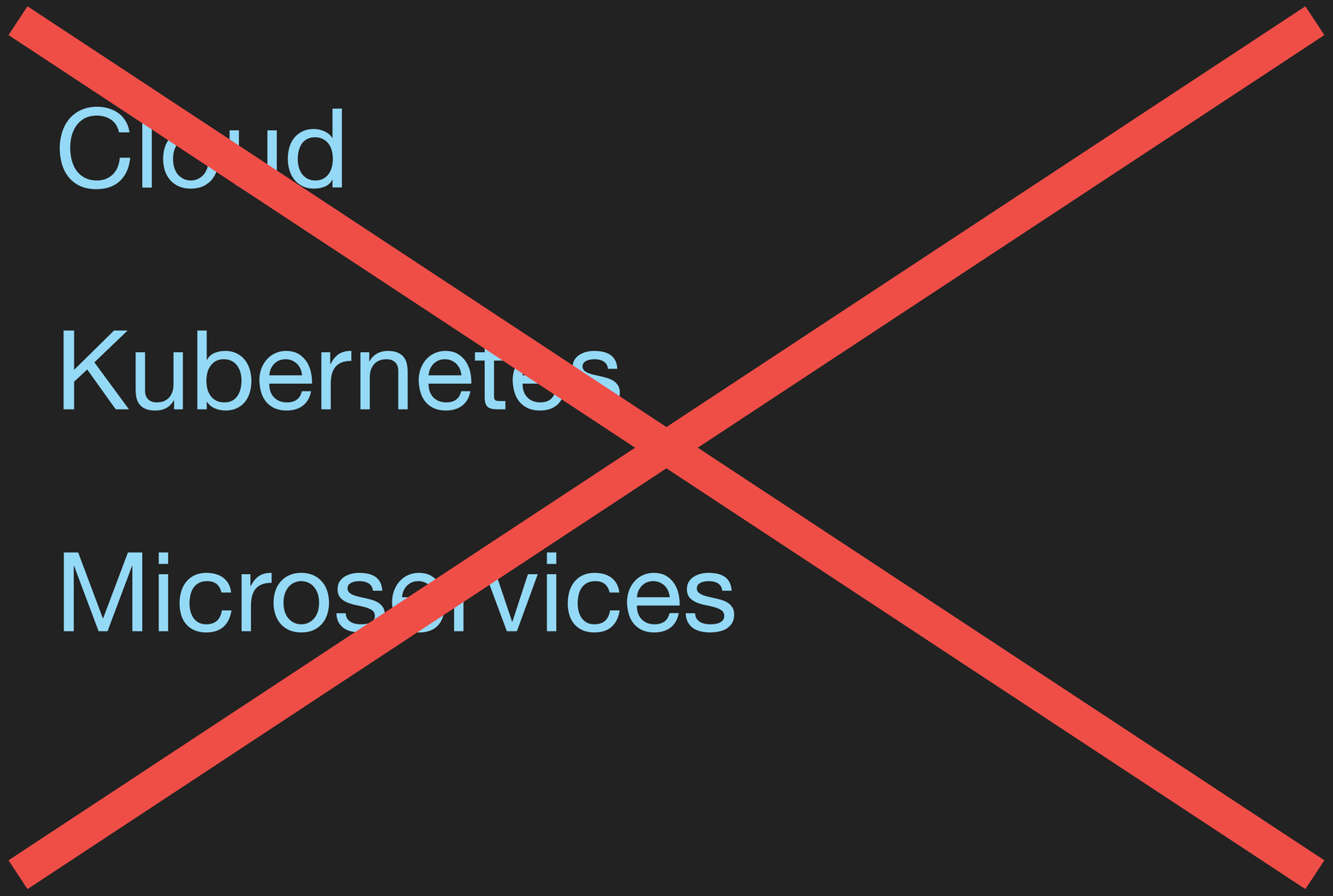
Consistency

Predictive planning

Cost reduction

These are not the outcomes you're looking for

ANTIPATTERN: “NEWER” EA OUTCOMES



These are not the outcomes you’re looking for

PATTERN: ALIGN TO BUSINESS AGILITY

Quality

Flow



Customers

Colleagues

Citizens

Climate

Security

Privacy

Minimum viable compliance

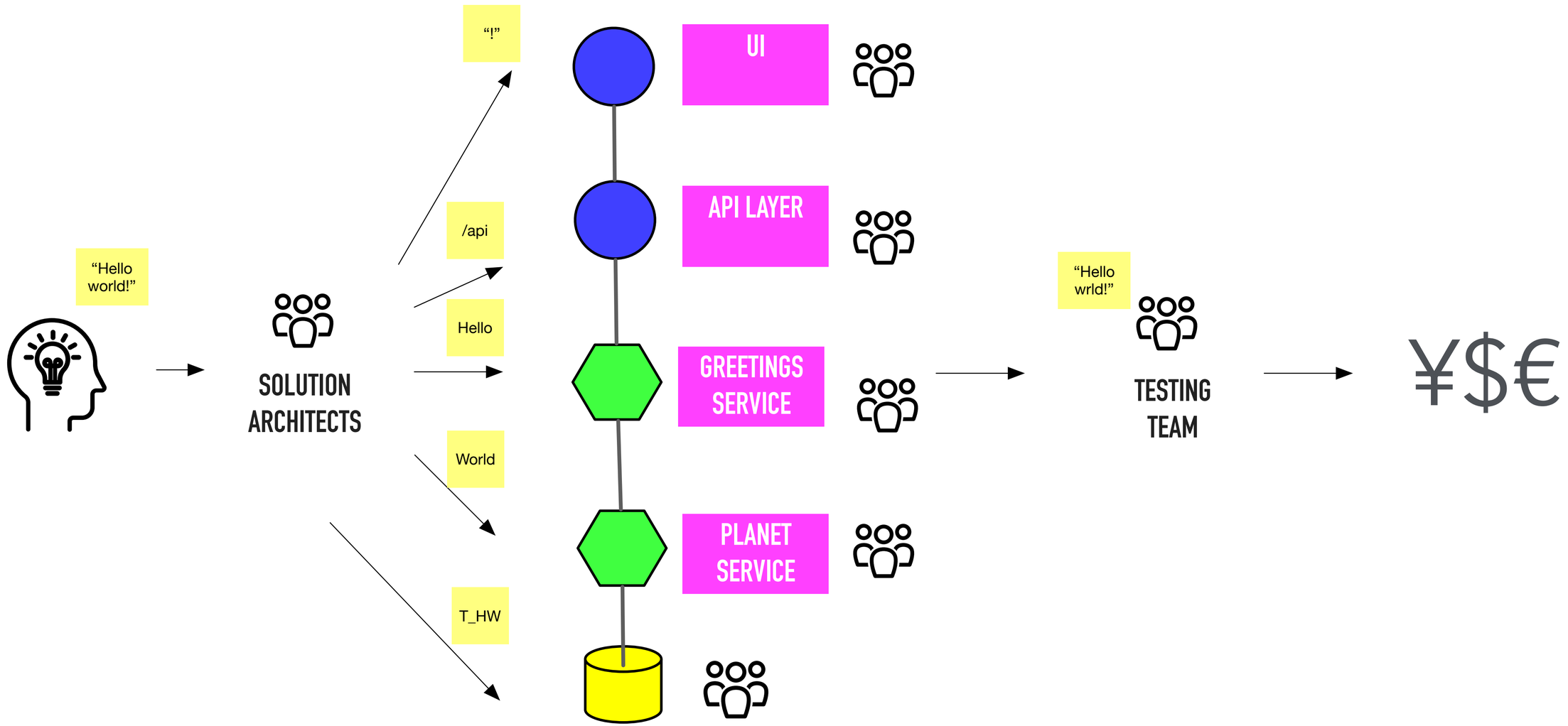
“WE SHAPE OUR ARCHITECTURE AND THEN THE ARCHITECTURE SHAPES US

AS THE SYSTEMS WE BUILD BECOME LARGER, THE COORDINATION INCREASES…

…IT CAN GROW SO SO LARGE THAT ALL OUR TIME AND ENERGY IS SPENT COORDINATING - IT IS AT THE EXPENSE OF OUR VALUE CREATING ACTIVITIES”

Gene Kim, DevOps Masterclass (2023)

“HELLO WORLD” WITH TIGHTLY COUPLED COMPLEX ARCHITECTURE





A beautiful layer cocktail

Layering (sometimes) Considered Harmful



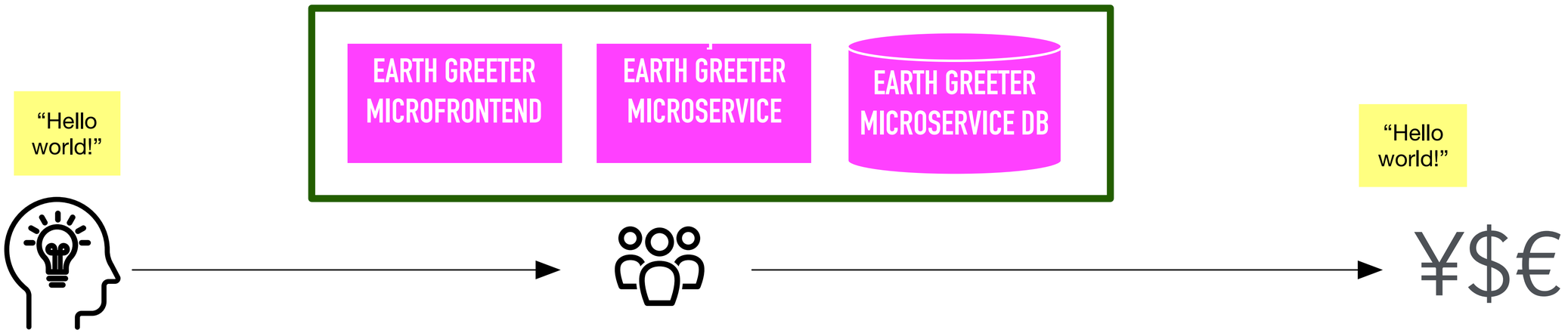
Gregor Hohpe

IT Strategist firmly entrenched in the cloud engine room. Author, Speaker, former Singapore Smart Nation Fellow

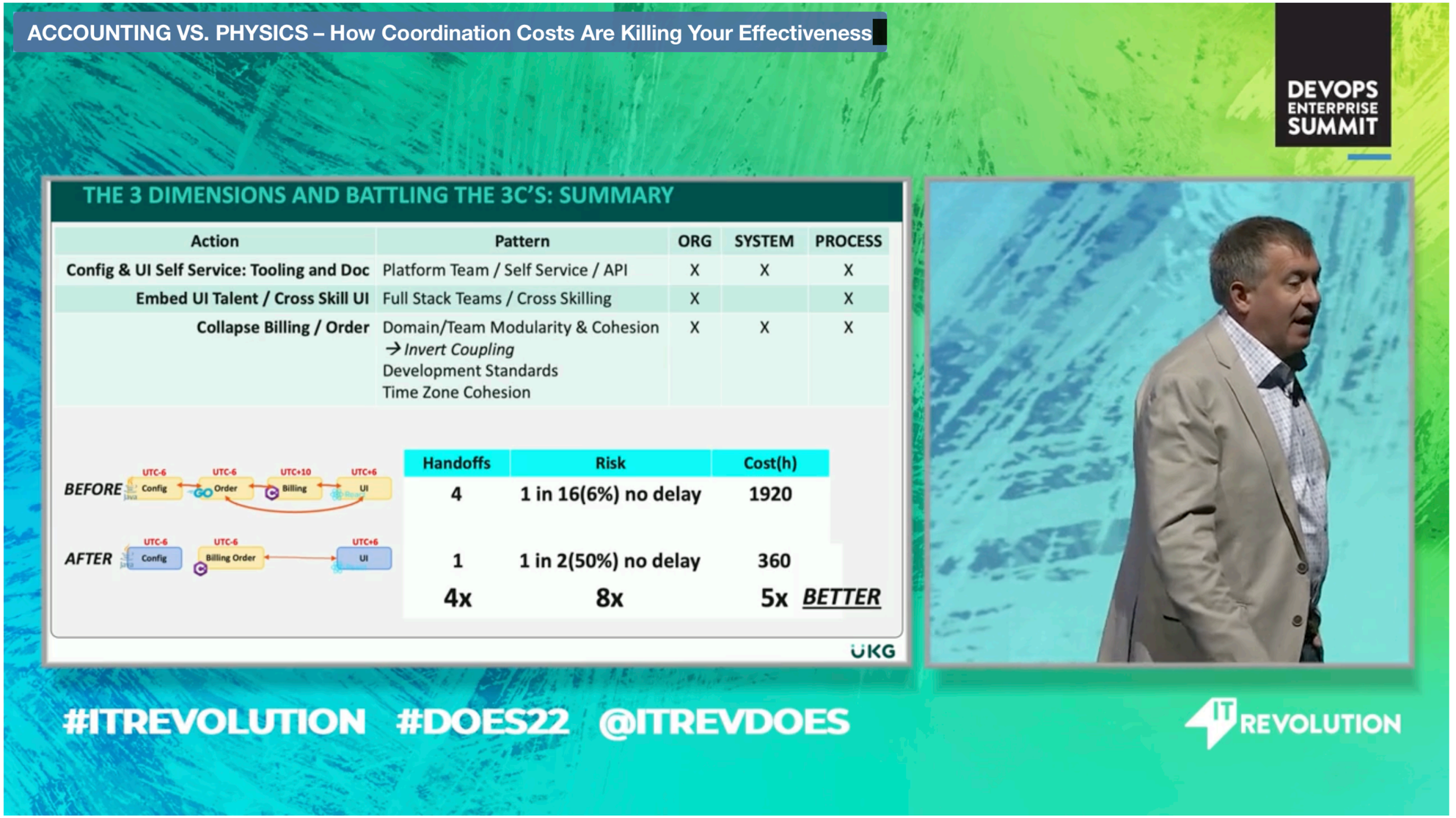
74 articles ✓ Following

November 13, 2016

AUTONOMOUS TEAM WITH THEIR OWN VALUABLE BUSINESS CAPABILITY



~~Refactored~~ Restructured organization and architecture



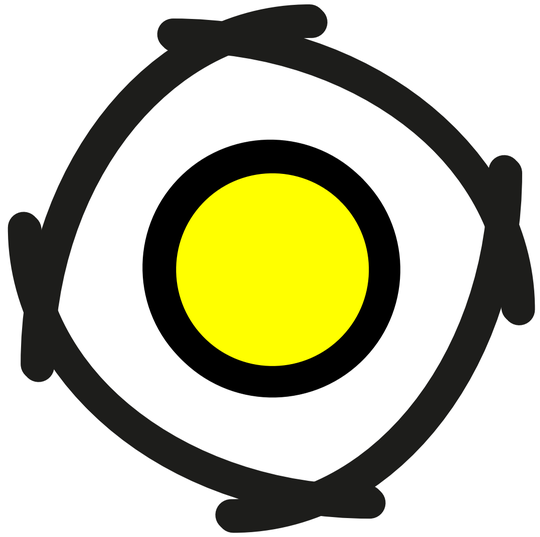
Las Vegas 2022

NO SLIDES AVAILABLE

ACCOUNTING VS. PHYSICS – How Coordination Costs Are Killing Your Effectiveness

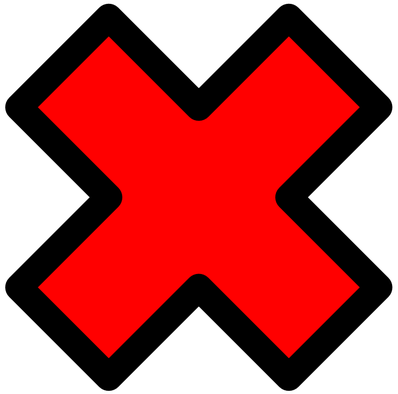
ACCOUNTING VS. PHYSICS – How Coordination Costs Are Killing Your Effectiveness

B



BETTER VALUE
SOONER SAFER
HAPPIER

RATHER THAN



DE-DUPLICATION
STANDARDISATION
COST REDUCTION

C



CONTINUOUS CONVERSATIONAL & AUTOMATED GOVERNANCE

“CREATING AND REDUCING COMPLEXITY MAY SOUND LIKE PERFECT OPPOSITES. BUT IN FACT FUNDAMENTAL ASYMMETRIES EXIST BETWEEN THE TWO.”

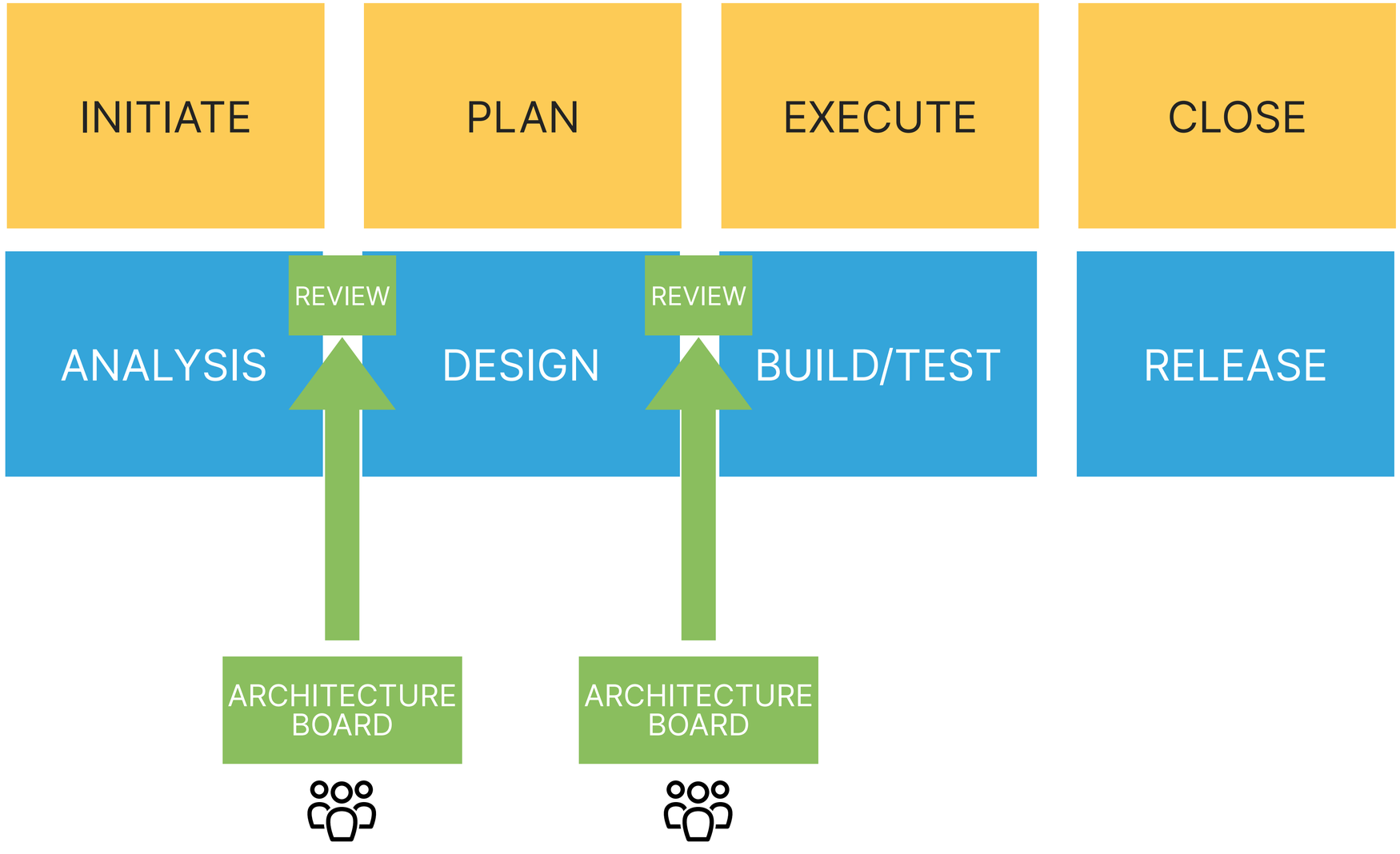
Reeves, Levin, Fink, Levina, Taming Complexity (2020), https://hbr.org/2020/01/taming-complexity

ARCHITECTURE GOVERNANCE TO THE RESCUE?

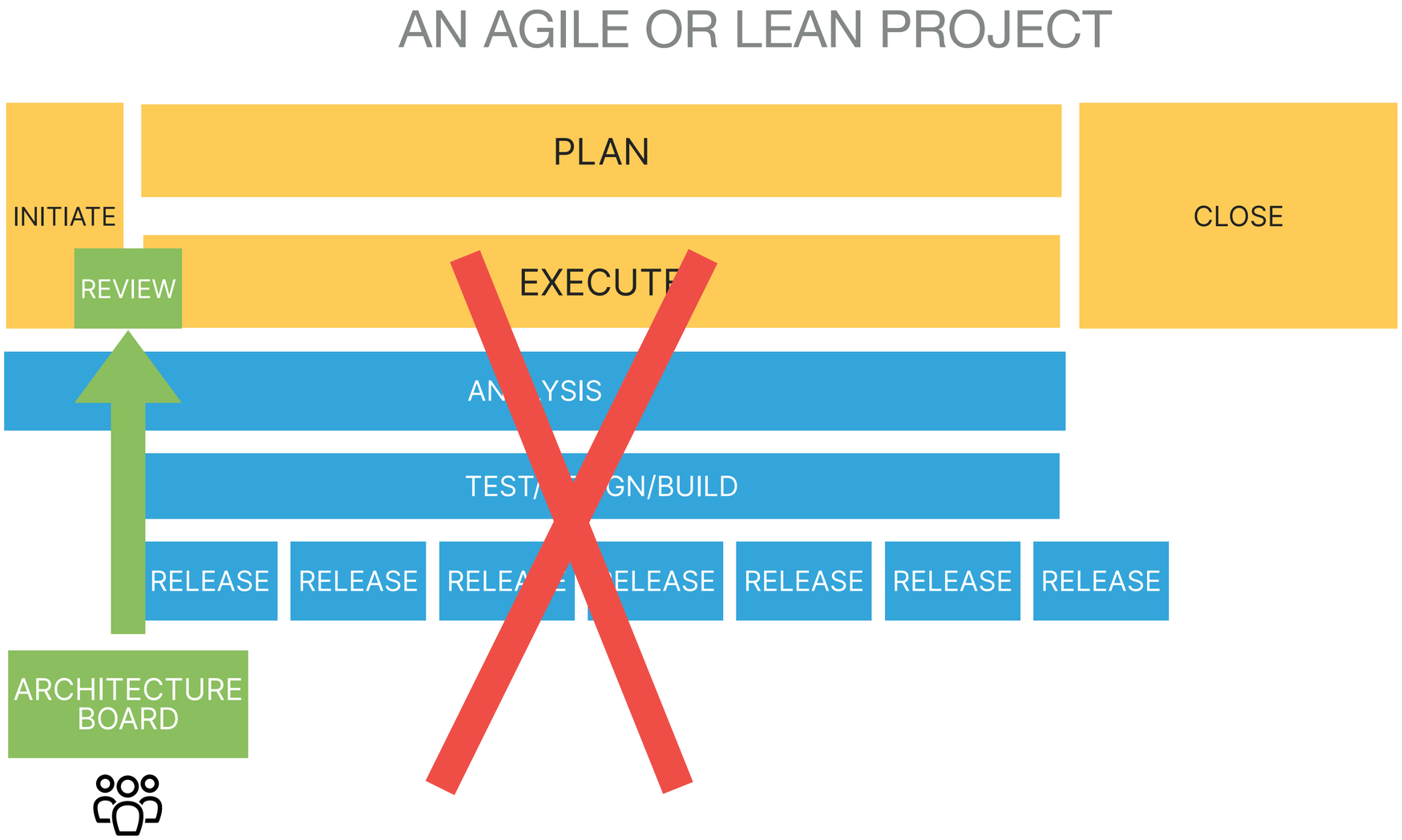
ANTIPATTERN: ARCHITECTURE REVIEW BOARDS

HOW WE USED TO EXECUTE ARCHITECTURE GOVERNANCE

A PROJECT: WATERFALL / WAGILE / WATER-SCRUM-FALL / “HYBRID”

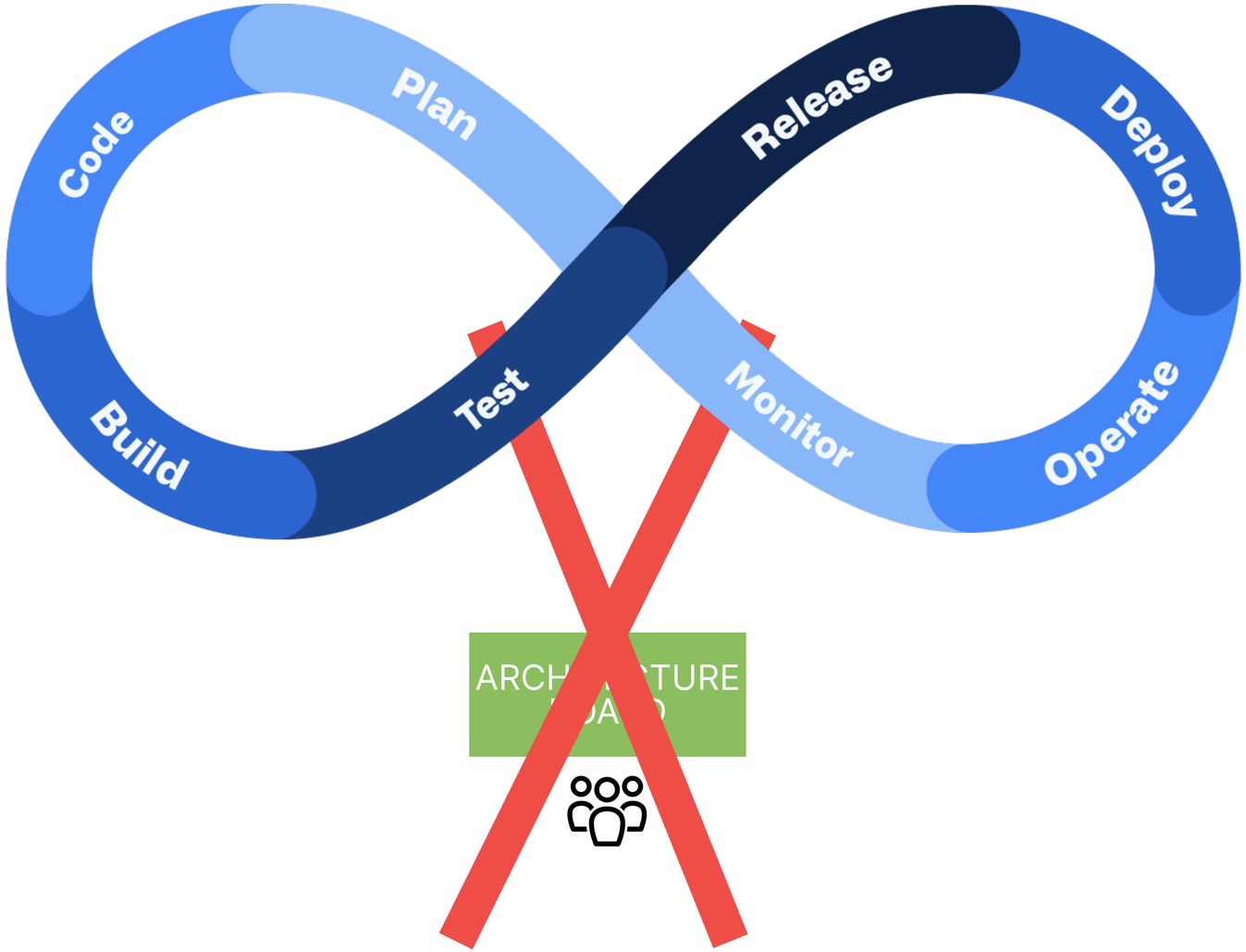


HOW DOES THIS WORK WITH MODERN DELIVERY?



HOW DOES THIS WORK WITH MODERN DELIVERY?

A PRODUCT



PATTERN:
BLACK BOX ARCHITECTURE GOVERNANCE

GOVERN REALISED ARCHITECTURES, NOT (JUST) DESIGNS ON PAPER

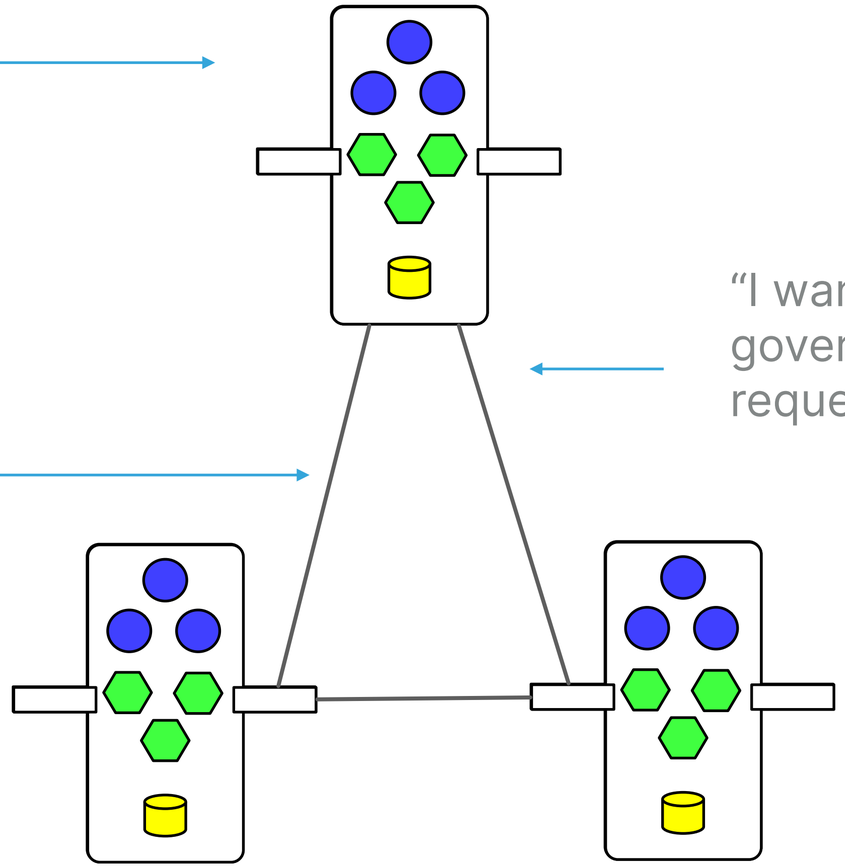
“I think I need an new service”

EA approve new services, new namespaces in Kubernetes (etc.)

“I think my service needs to access data or functionality from another service”

EA approve:

- Service-to-service IAM
- Firewalls



“I want an exception to automated governance - e.g. synchronous request/response”

EA approve:

- Exceptions
- Tech debt

Paved paths for quality, security and flow

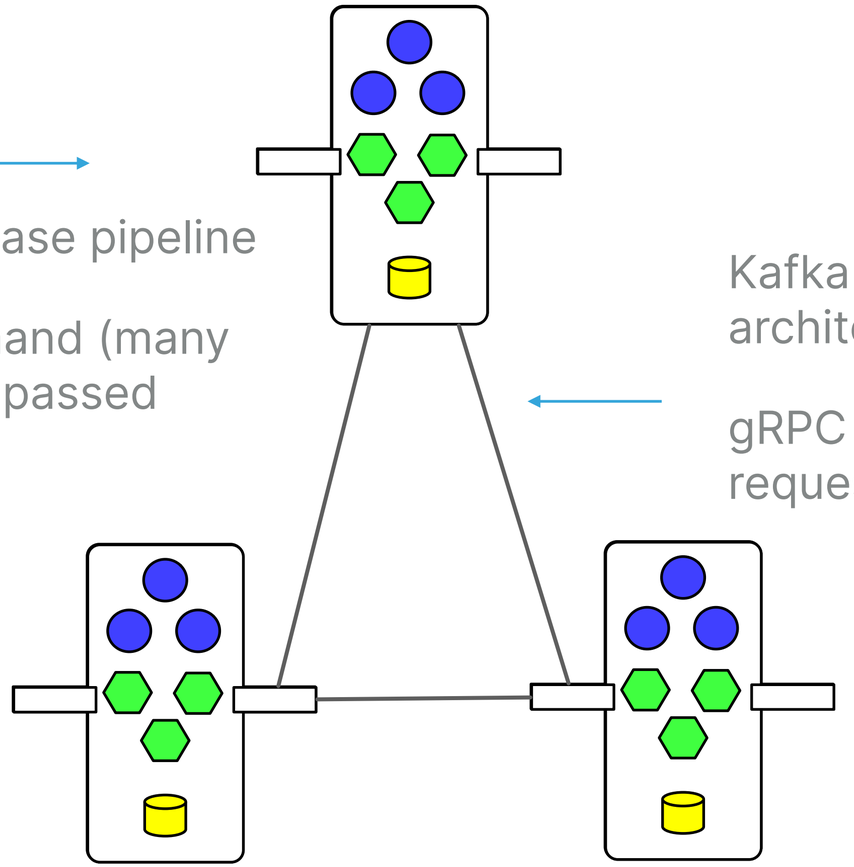
Azure Kubernetes Service /

Docker /

.NET or Python /

Scaffolded code + build + release pipeline

Release to production on-demand (many times per day) if quality gates passed

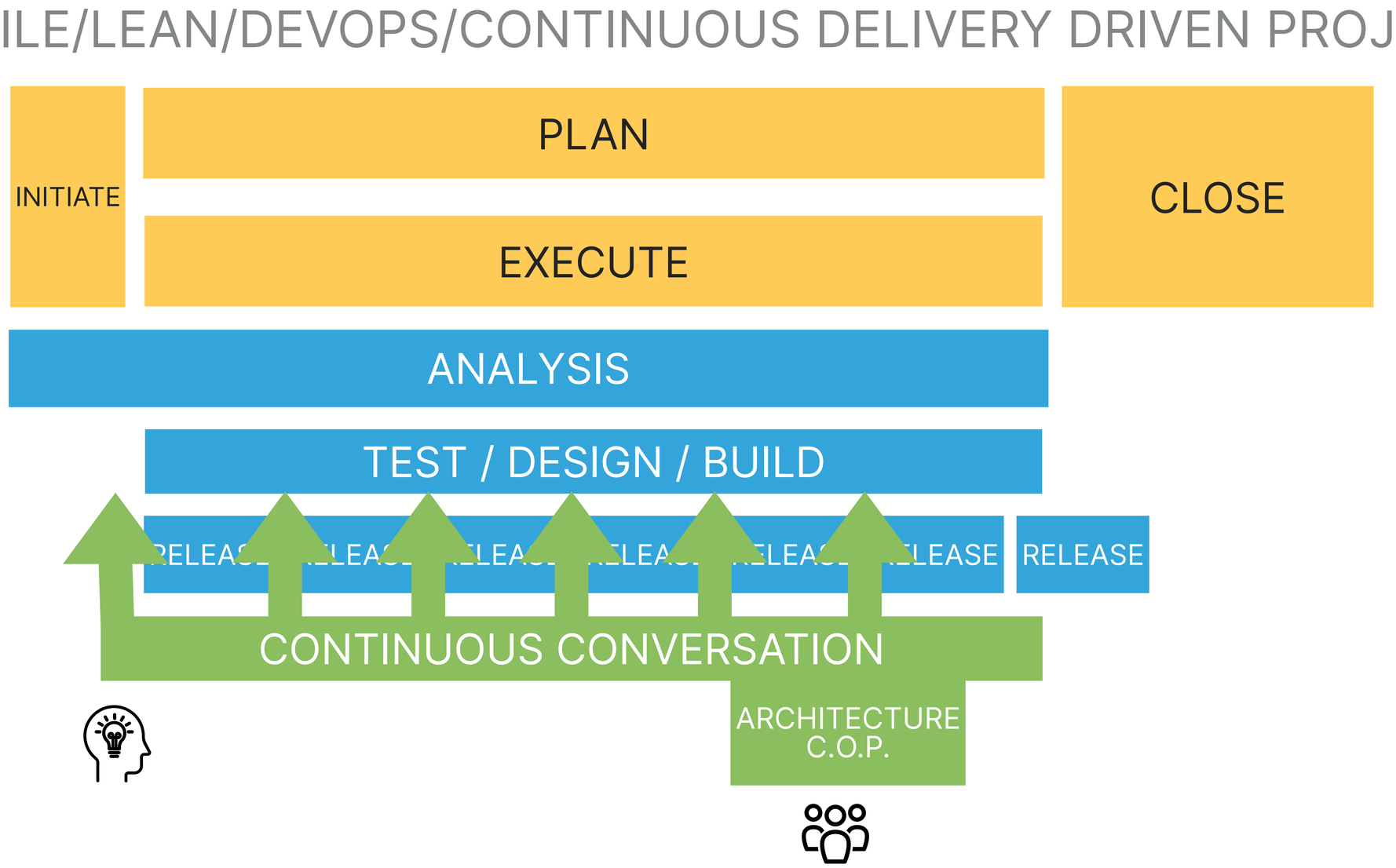


Kafka for event driven architecture + data push

gRPC by exception for sync request/response

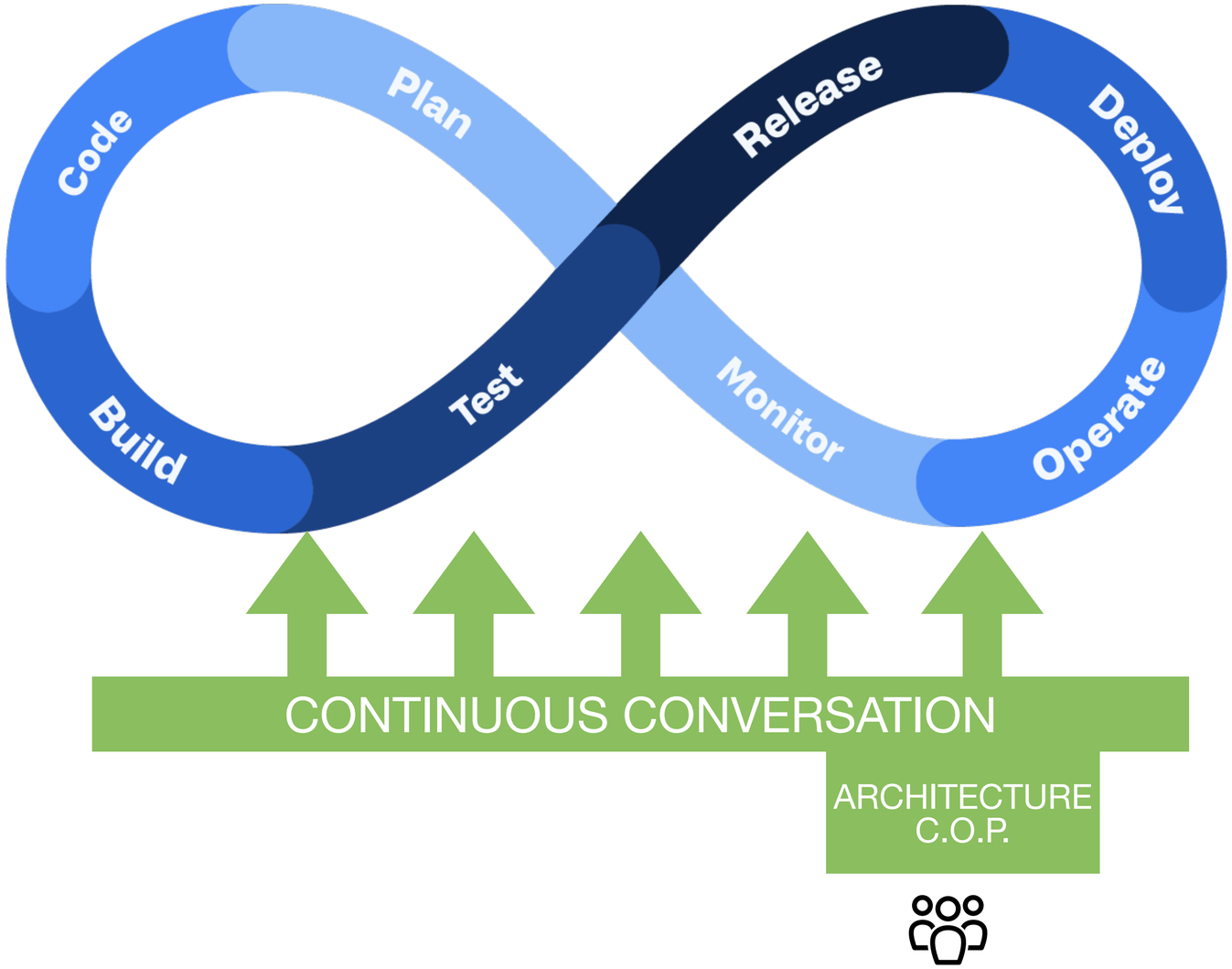
PATTERN:
CONTINUOUS CONVERSATIONAL “GOVERNANCE”

CONTINUOUS CONVERSATIONAL GOVERNANCE

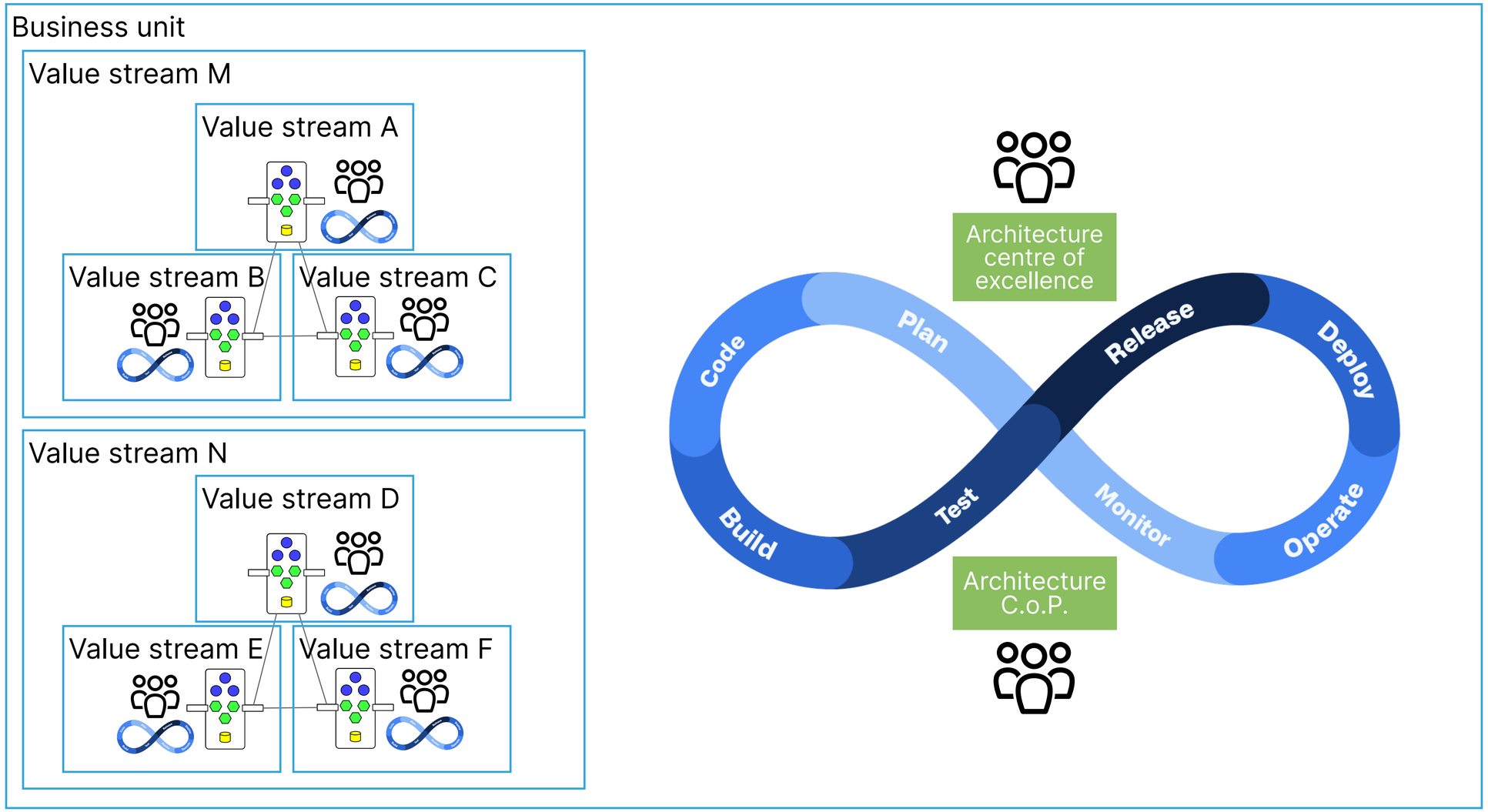


CONTINUOUS CONVERSATIONAL GOVERNANCE

A SINGLE PRODUCT



CONTINUOUS CONVERSATIONAL GOVERNANCE ACROSS VALUE STREAMS



PATTERN:
PLATFORM ENGINEERING AS AUTOMATED GOVERNANCE & COMPLIANCE

“ENTERPRISE ARCHITECTURE WAS OFTEN TOO HIGH-HANDED AND BUREAUCRATIC TO SUPPORT RAPID INNOVATION

BUT SOUND ARCHITECTURAL PRINCIPLES AND AGILE PROCESSES AREN’T MUTUALLY EXCLUSIVE

IN A PROPERLY EXECUTED PLATFORM STRATEGY, MAJOR ARCHITECTURE AND SECURITY CHECKS HAVE HAPPENED, THAT 50-ITEM CHECKLIST DIMINISHES BY 90%.”

Charles Betz, Businesses elevate IT architecture to fight back tech sprawl (2024)

“THE PATH FORWARD IS A TWO-WAY STREET, WITH STAKEHOLDERS ACCEPTING A LEVEL OF STANDARDIZATION AND ARCHITECTS NOT STANDING IN THE WAY OF PROGRESS.”

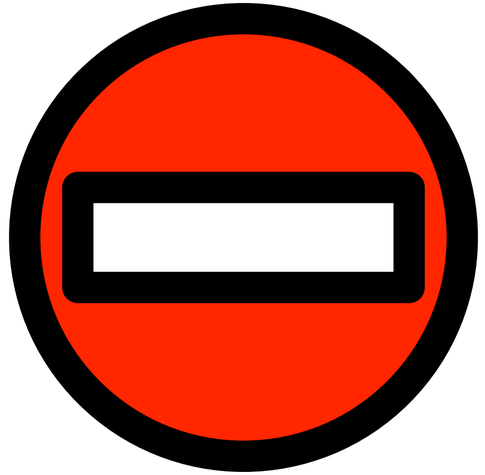
Charles Betz, Businesses elevate IT architecture to fight back tech sprawl (2024)

C

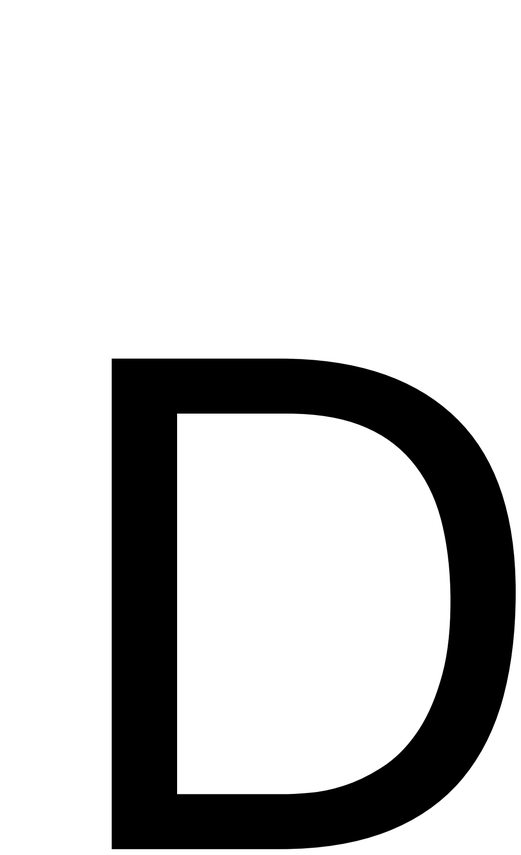


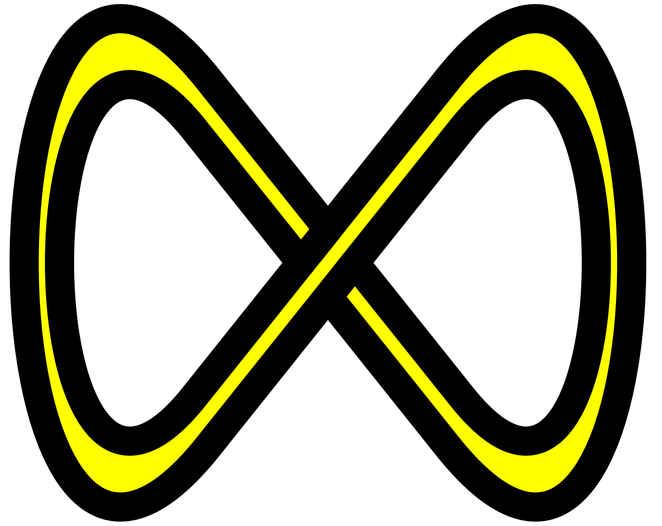
CONTINUOUS
CONVERSATIONAL
& AUTOMATED
GOVERNANCE

RATHER THAN



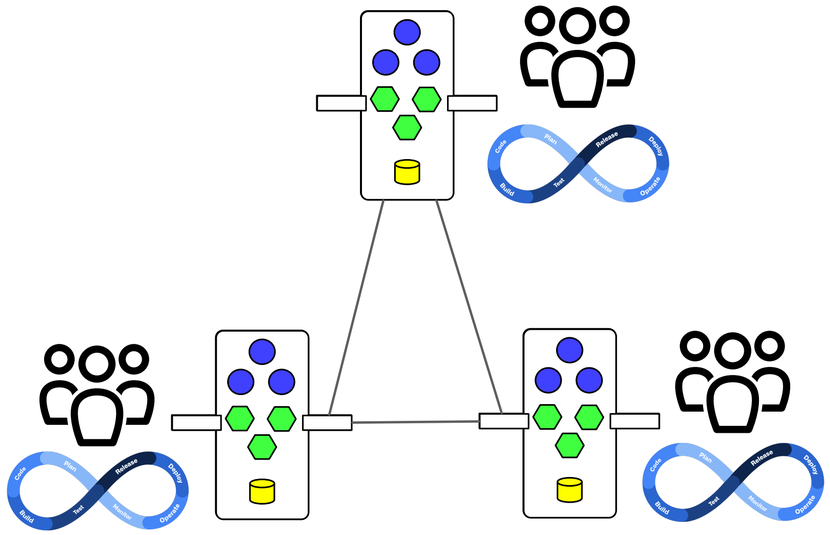
PHASE-GATED
DOCUMENT-LED
ADVERSARIAL
GOVERNANCE

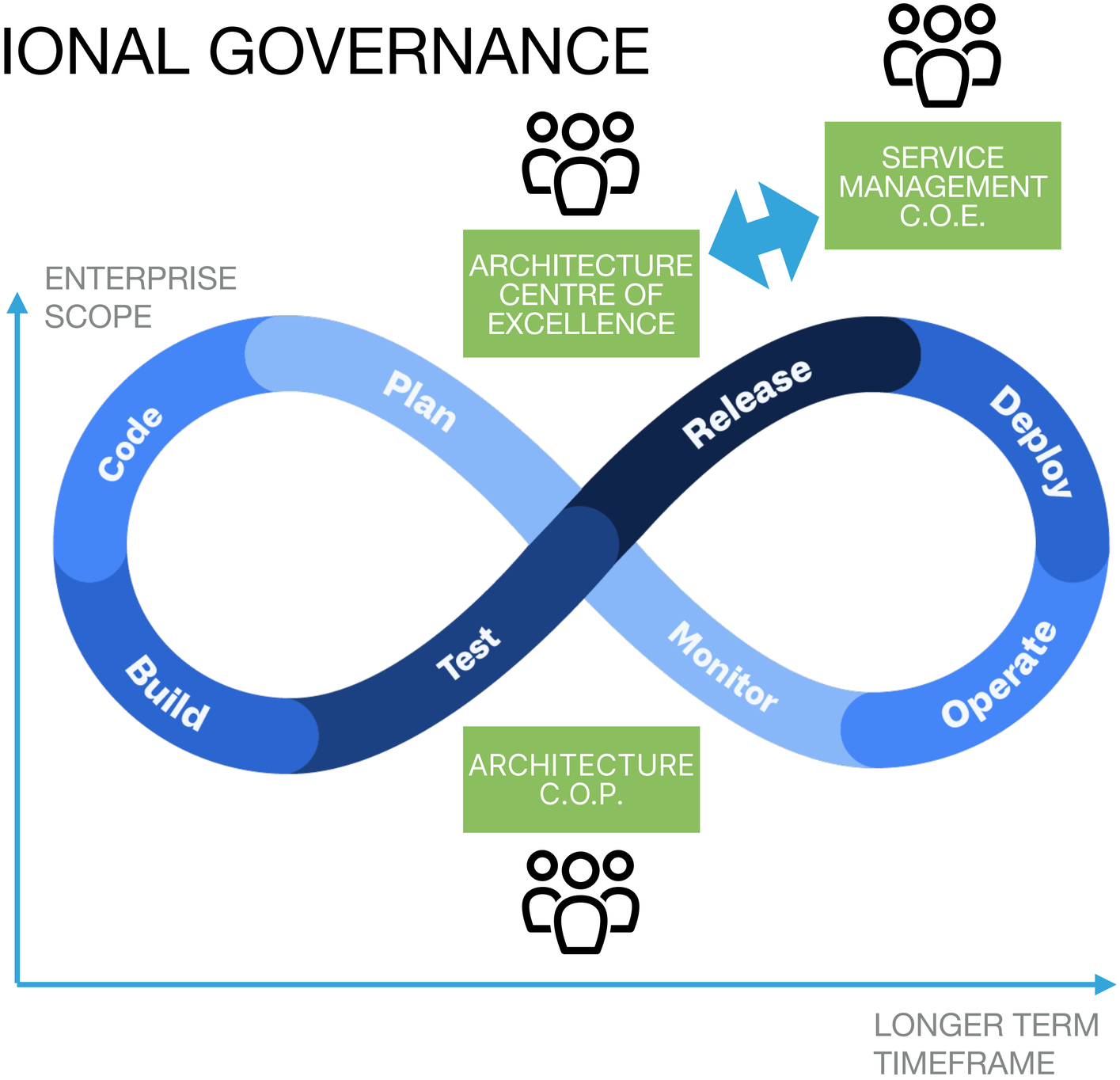




DEVOPS AT ENTERPRISE SCALE

CONTINUOUS CONVERSATIONAL GOVERNANCE

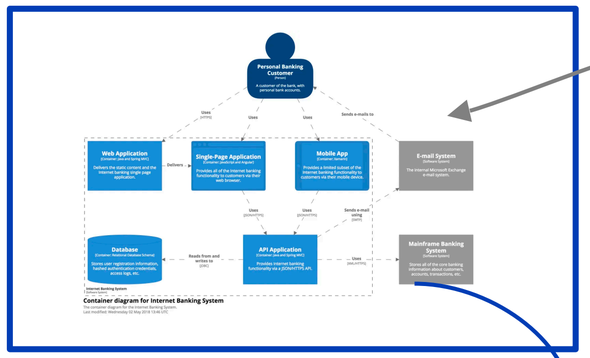


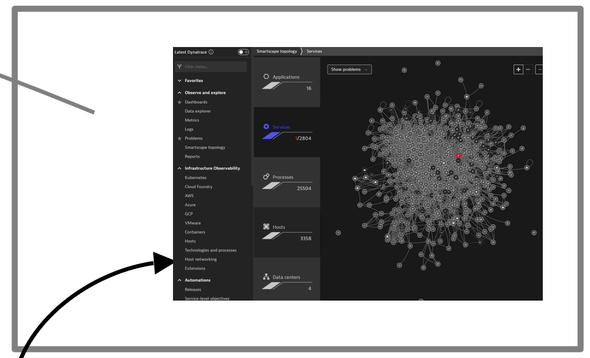


CONTINUOUS CONVERSATIONAL GOVERNANCE (AFTER RUTH MALAN)

INTENTION

REFLECTION





Whiteboard sessions

Model storming

Continuous refactoring

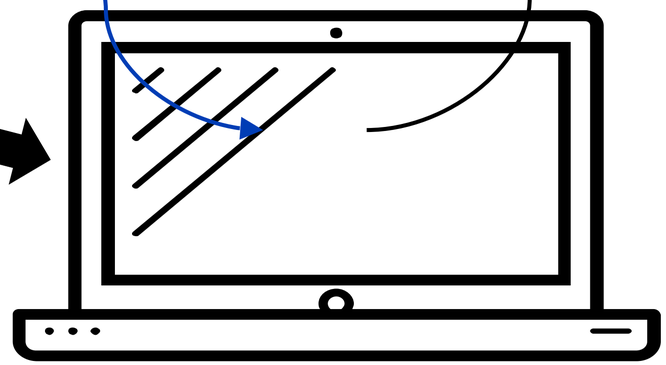
Test-driven design

Architecture decision records

Emergent design

Discoverability & observability tooling

Infrastructure-as-code



WORKING CODE DEPLOYED AS RUNNING SYSTEM

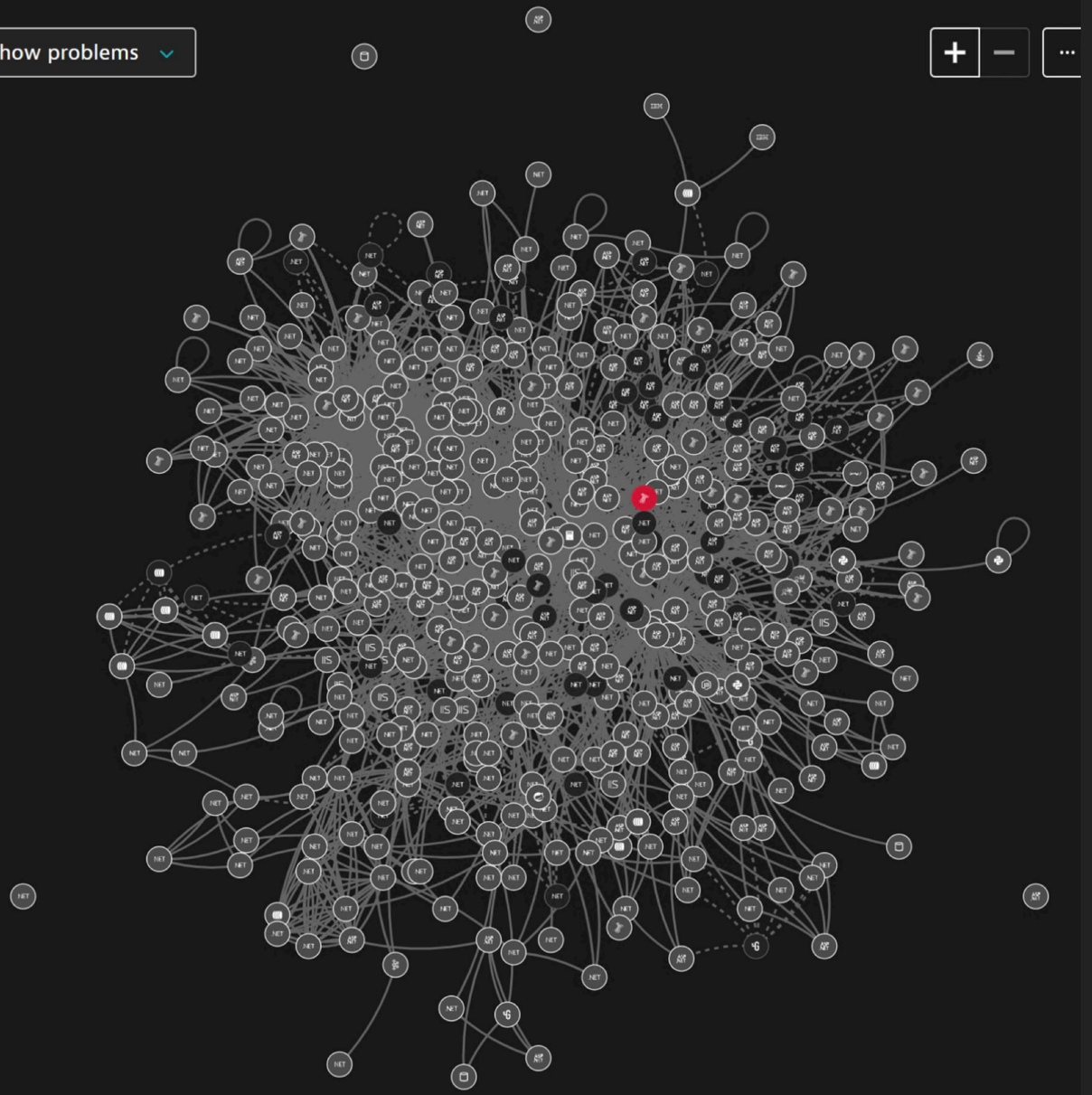
Applications
16

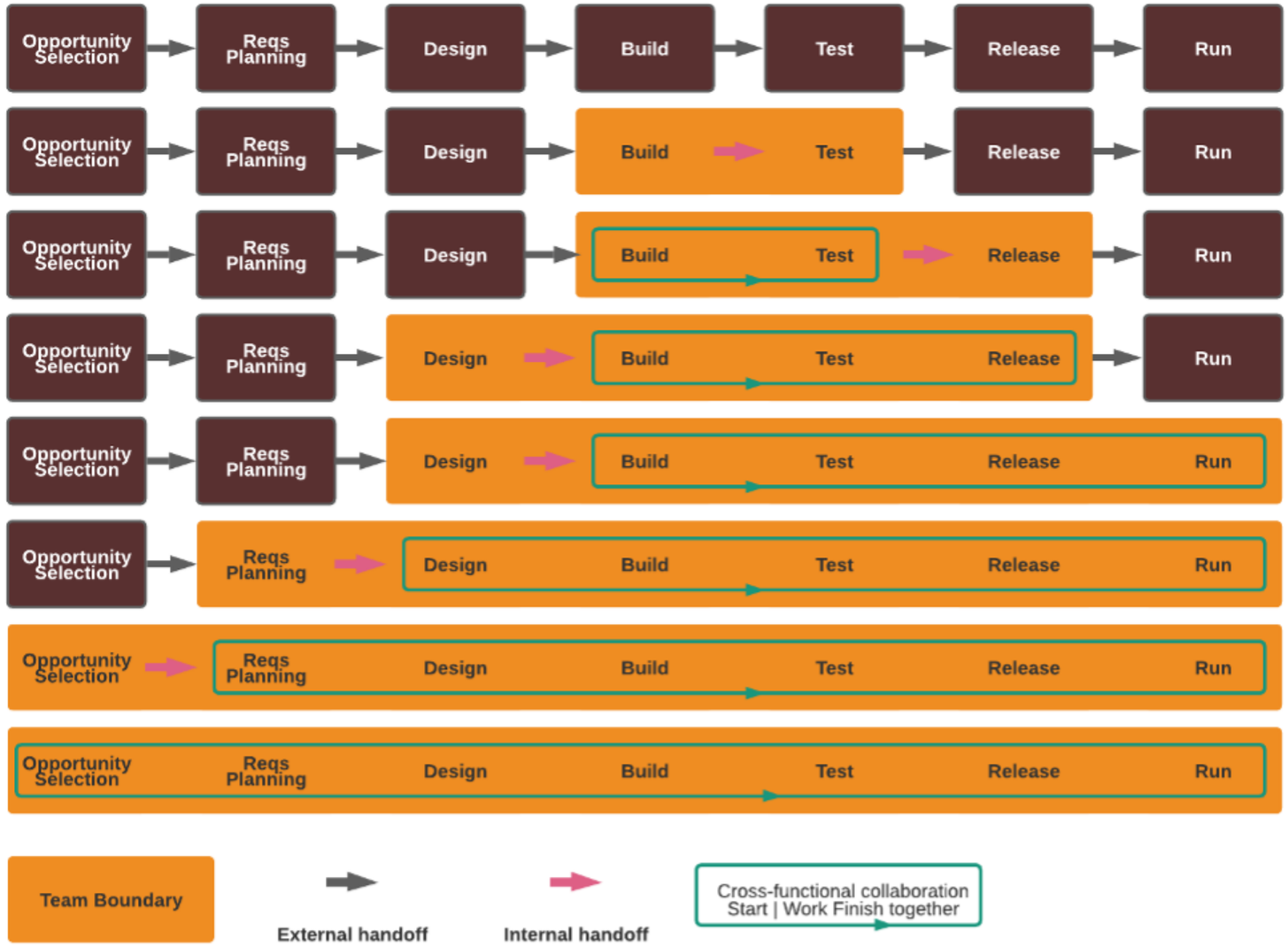
Services
1/2804

Processes
25504

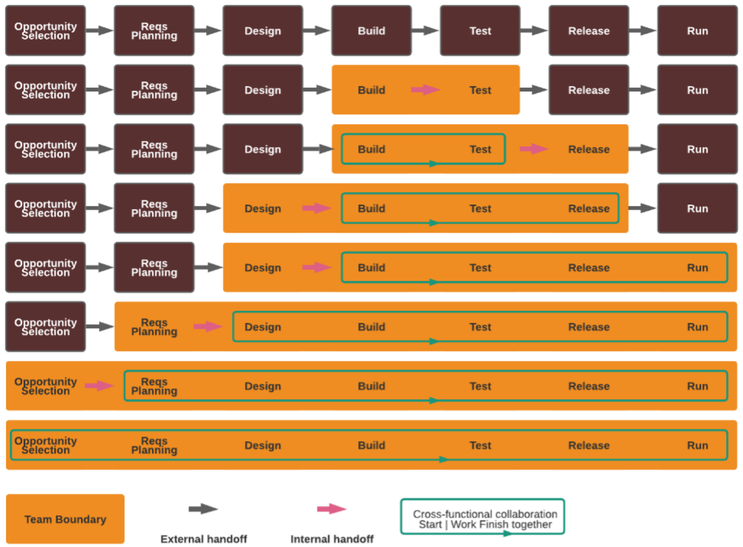
Hosts
3358

Data centers
4

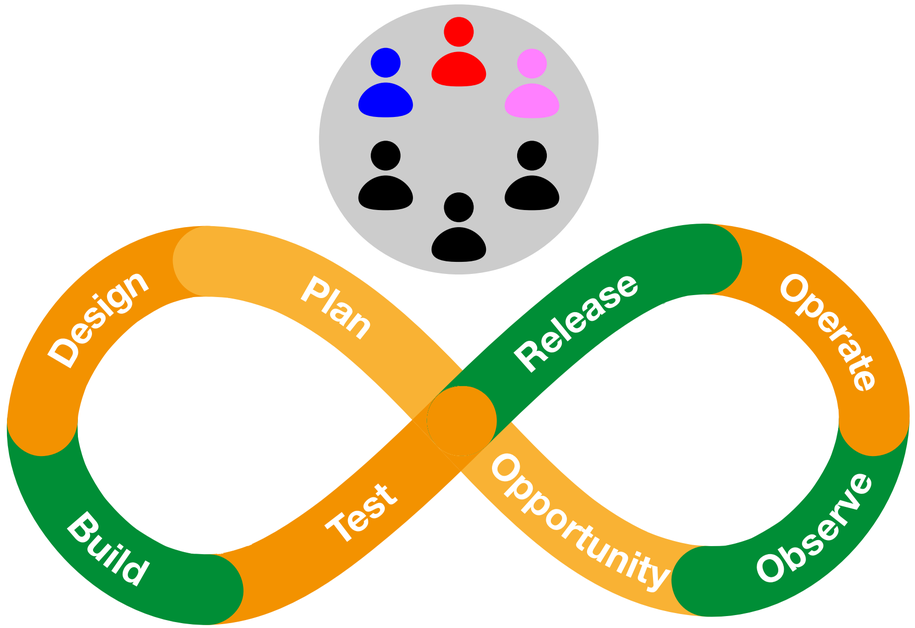




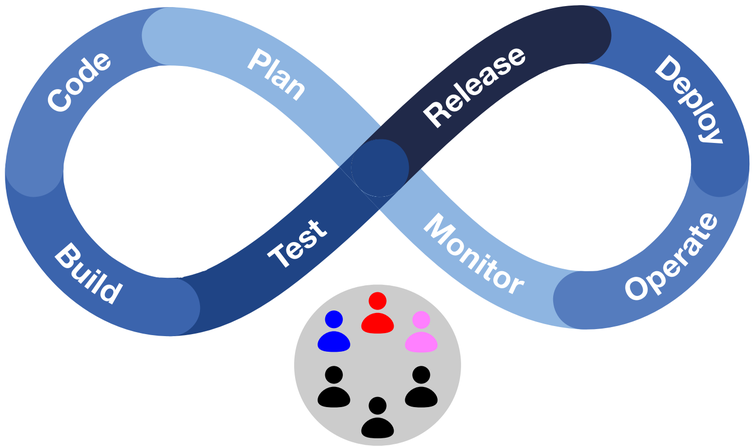
John Cutler, https://www.linkedin.com/posts/johnpcutler_im-working-on-an-addition-to-this-thought-activity-7301284507452612608-2Qm5 (2025)

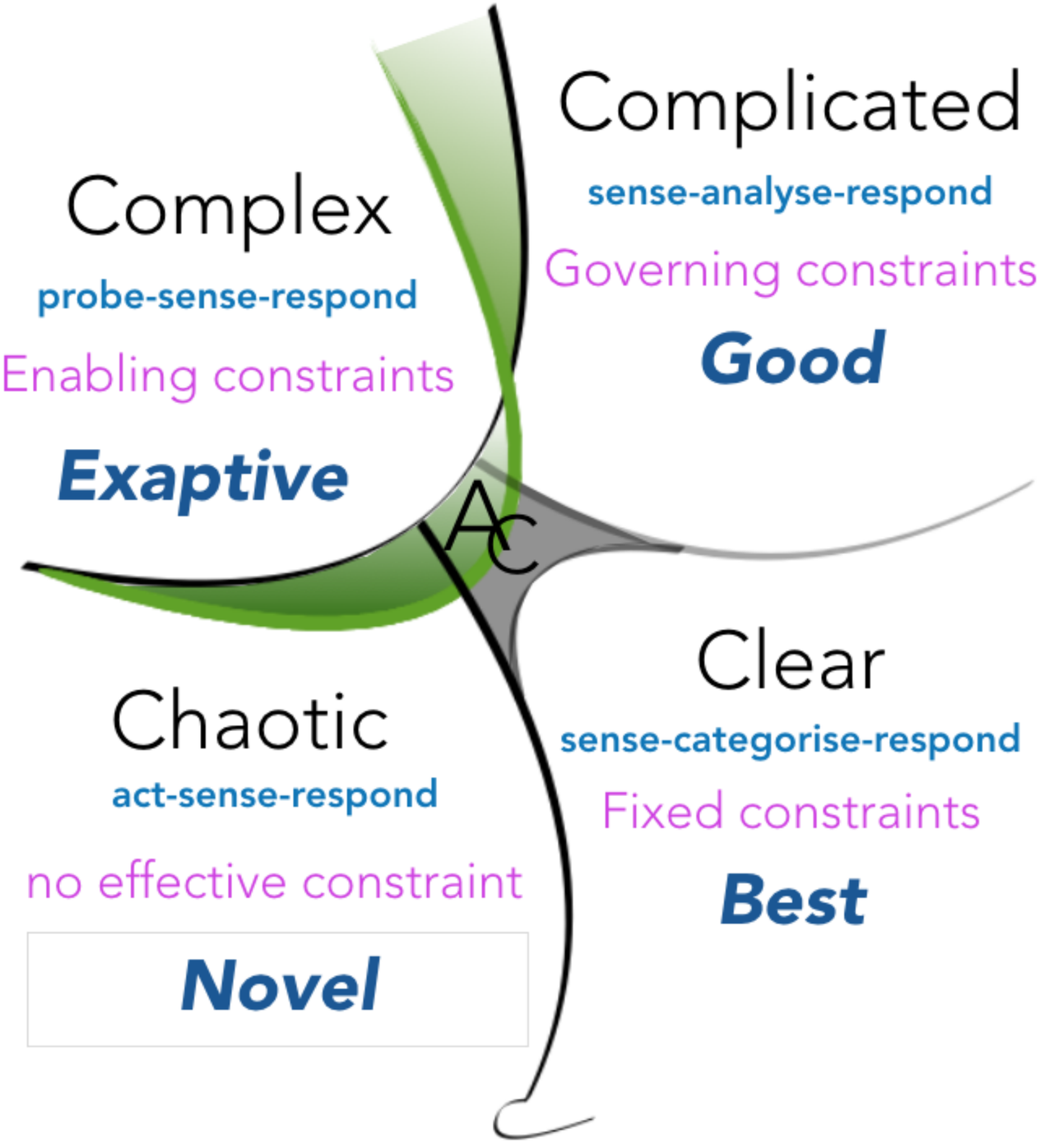

Opportunity Selection
Reqs Planning
Design
Build
Test
Release
Run
Team Boundary
External handoff
Internal handoff
Cross-functional collaboration
Start | Work Finish together

=


Design
Plan
Release
Operate
Build
Test
Opportunity
Observe

+


Code
Plan
Release
Deploy
Build
Test
Monitor
Operate


Complicated
sense-analyse-respond
Governing constraints
Good
Complex
probe-sense-respond
Enabling constraints
Exaptive
AC
Clear
sense-categorise-respond
Fixed constraints
Best
Chaotic
act-sense-respond
no effective constraint
Novel

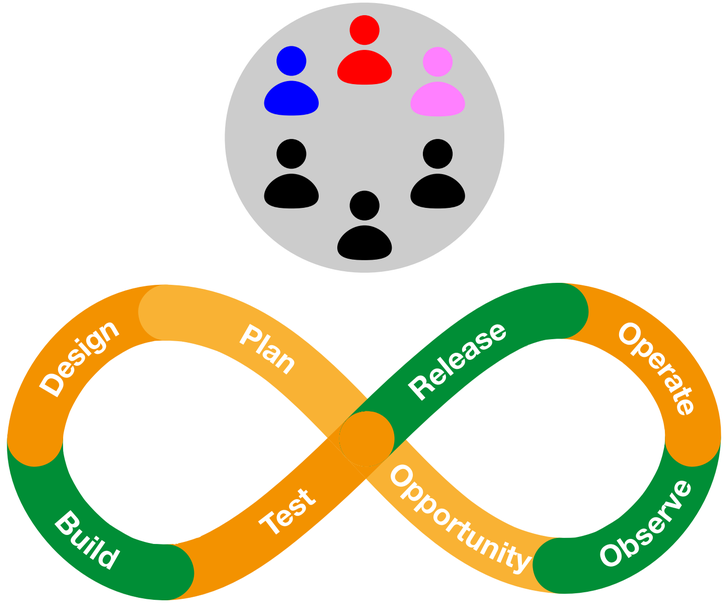
YOU

DEV

BUILD

CHANGE

IT



YOU

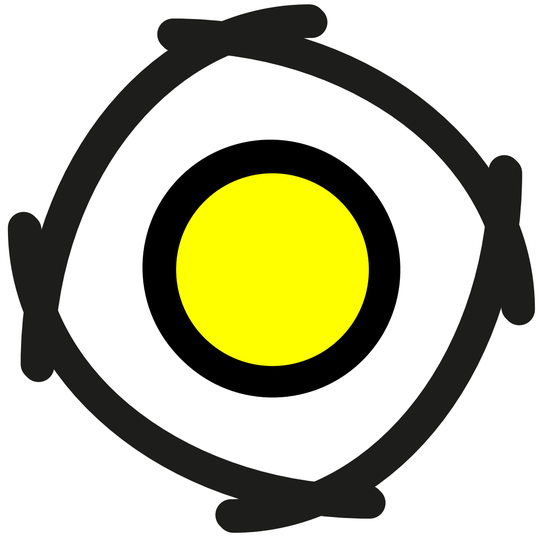
OPS

RUN

OPERATE

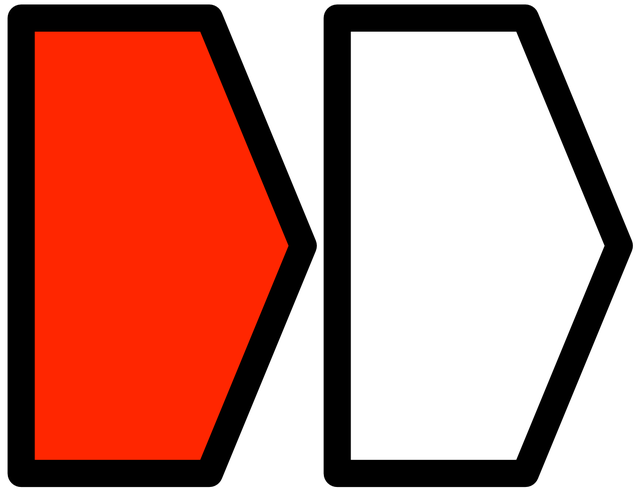
IT

D



DEVOPS AT ENTERPRISE SCALE

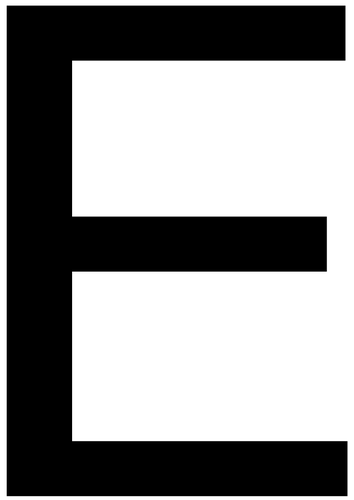
RATHER THAN

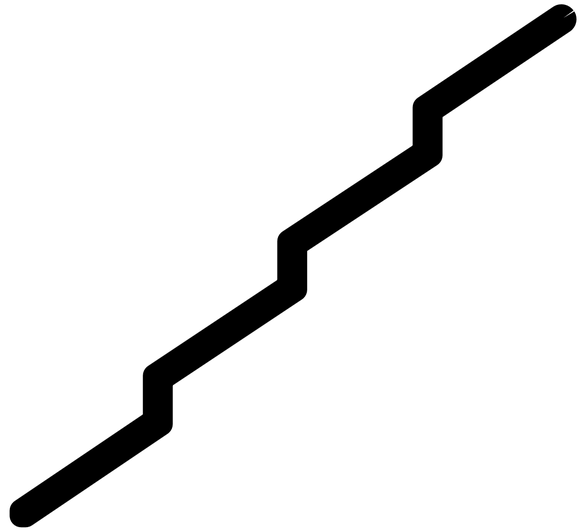


CHANGE NOT RUN

DEV NOT OPS

PROJECT NOT PRODUCT





EVOLUTIONARY ENTERPRISE ARCHITECTURE

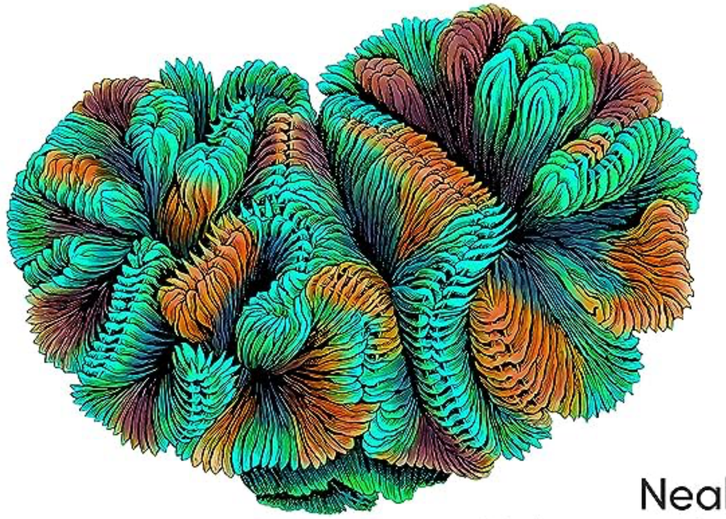
CURATING THE EVOLUTION OF ENTERPRISE ARCHITECTURE AND ORGANISATION

O'REILLY®

2nd Edition

Building Evolutionary Architectures

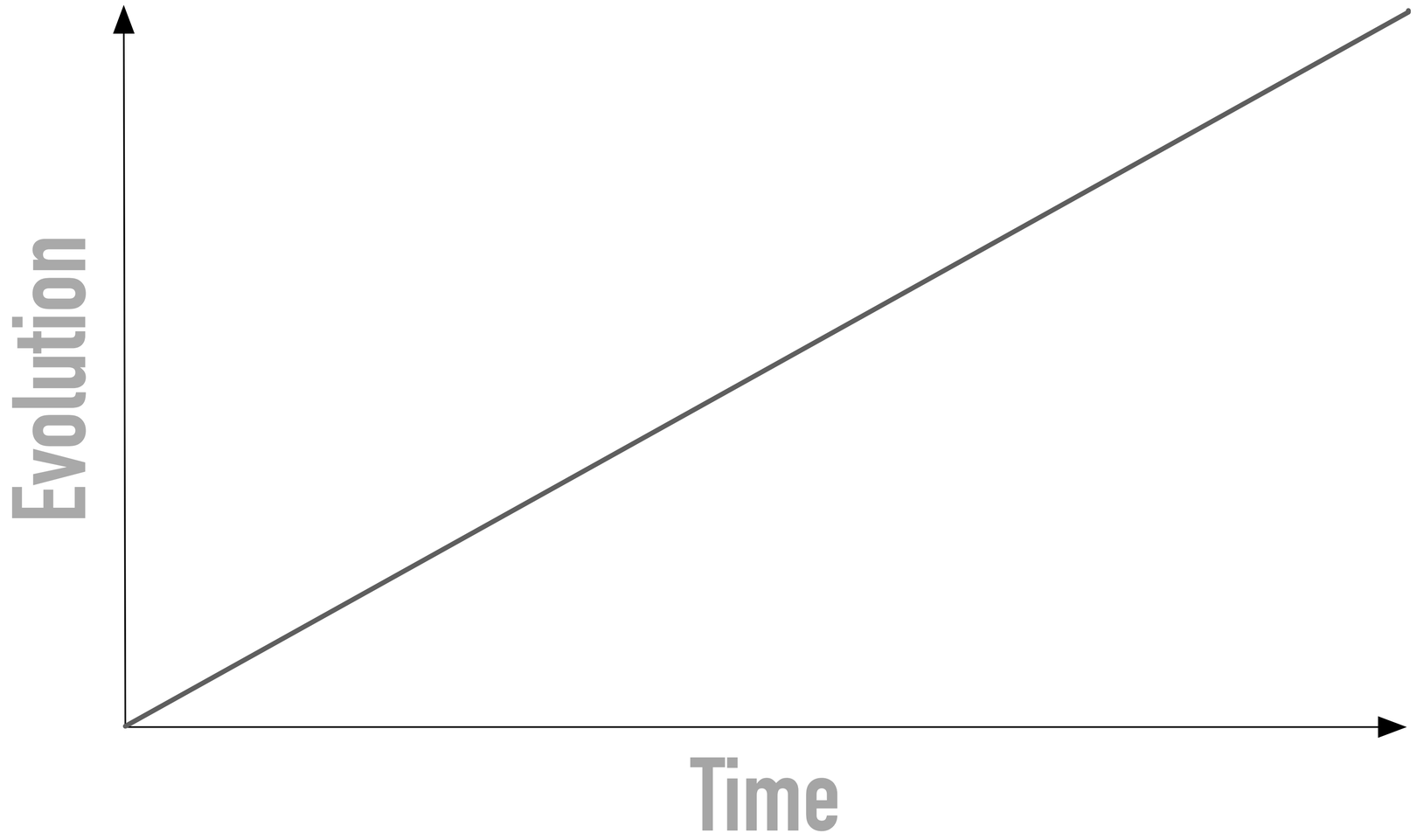
Automated Software Governance



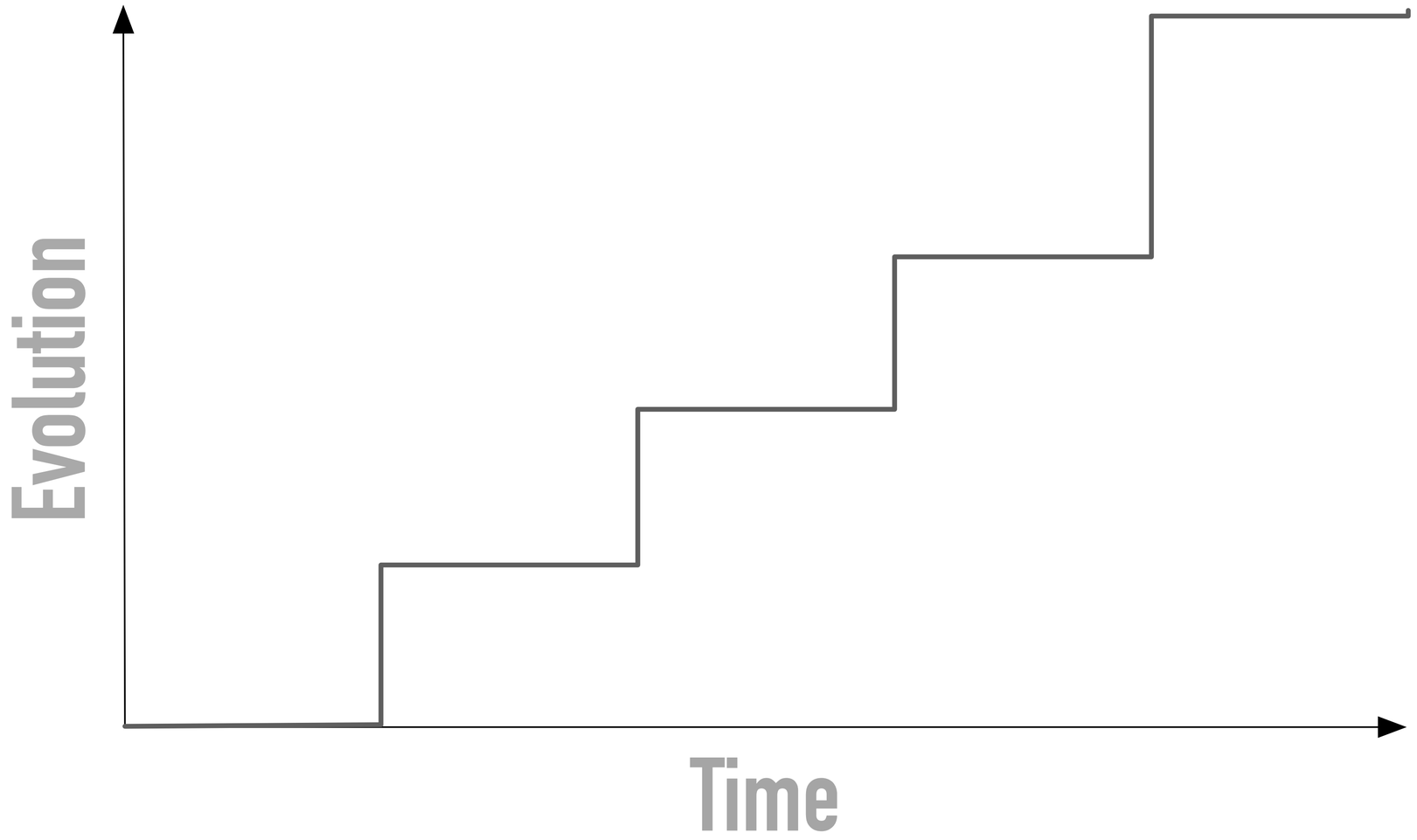
Neal Ford,
Rebecca Parsons,
Patrick Kua & Pramod Sadalage

Forewords by Mark Richards & Martin Fowler

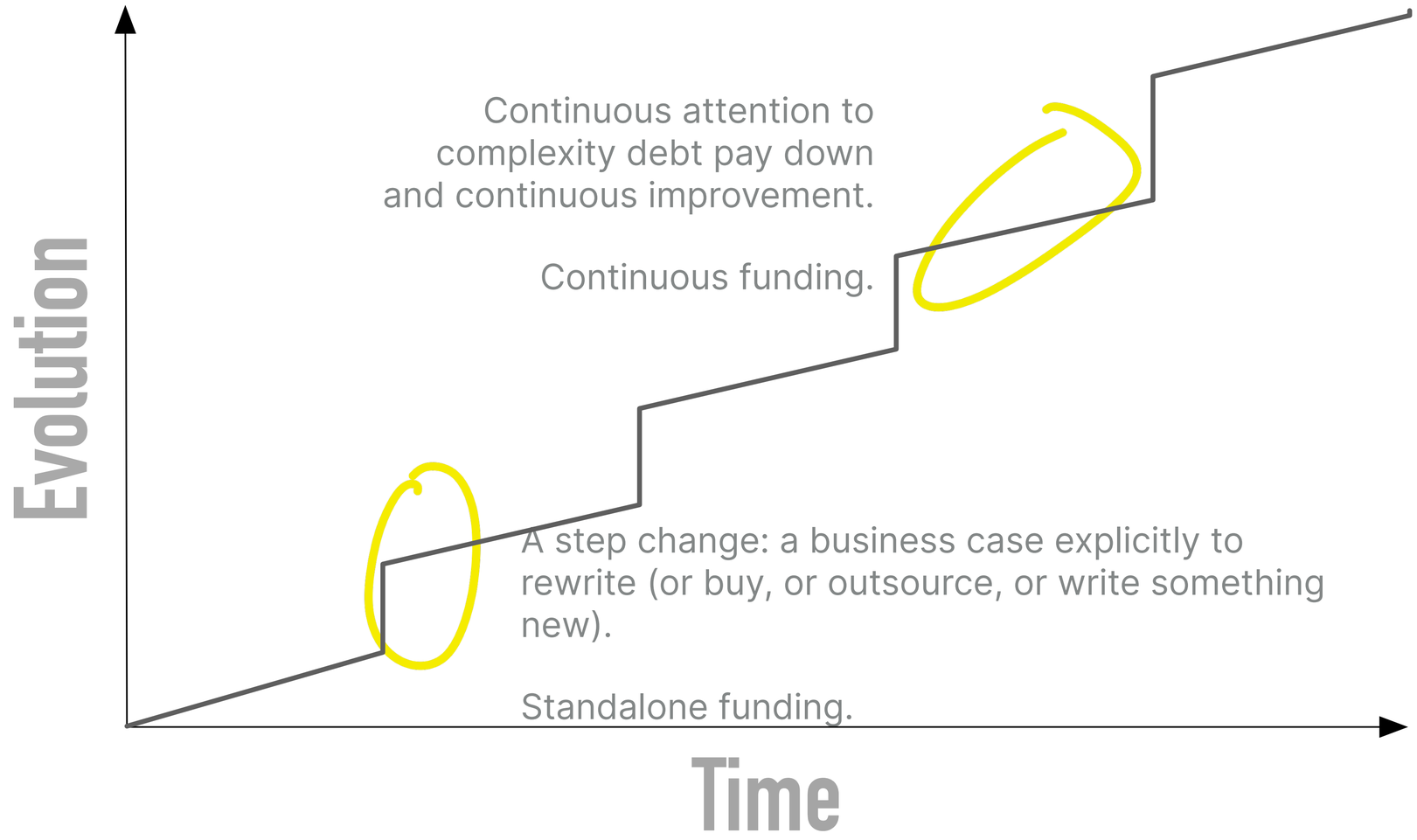
THEORY #1: GRADUAL EVOLUTION



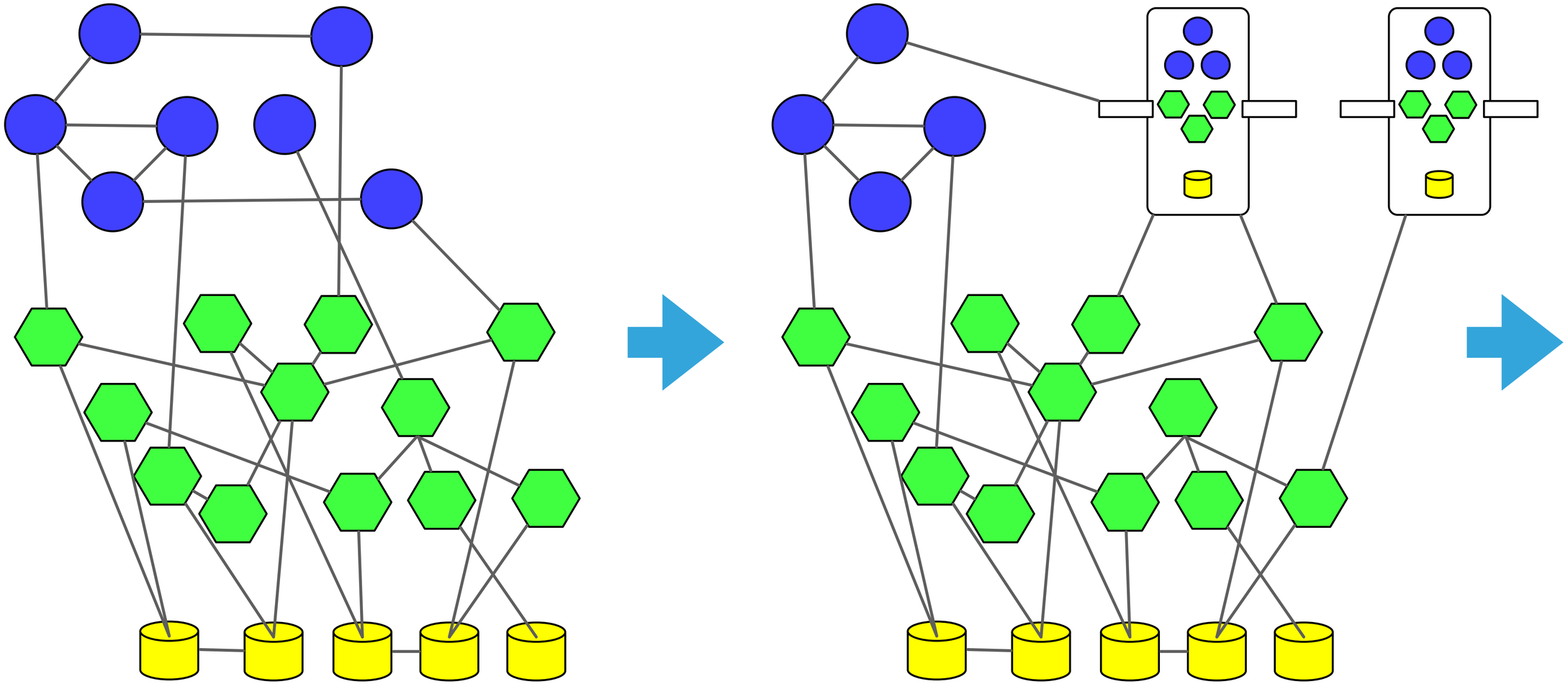
THEORY #2: PUNCTUATED EQUILIBRIA

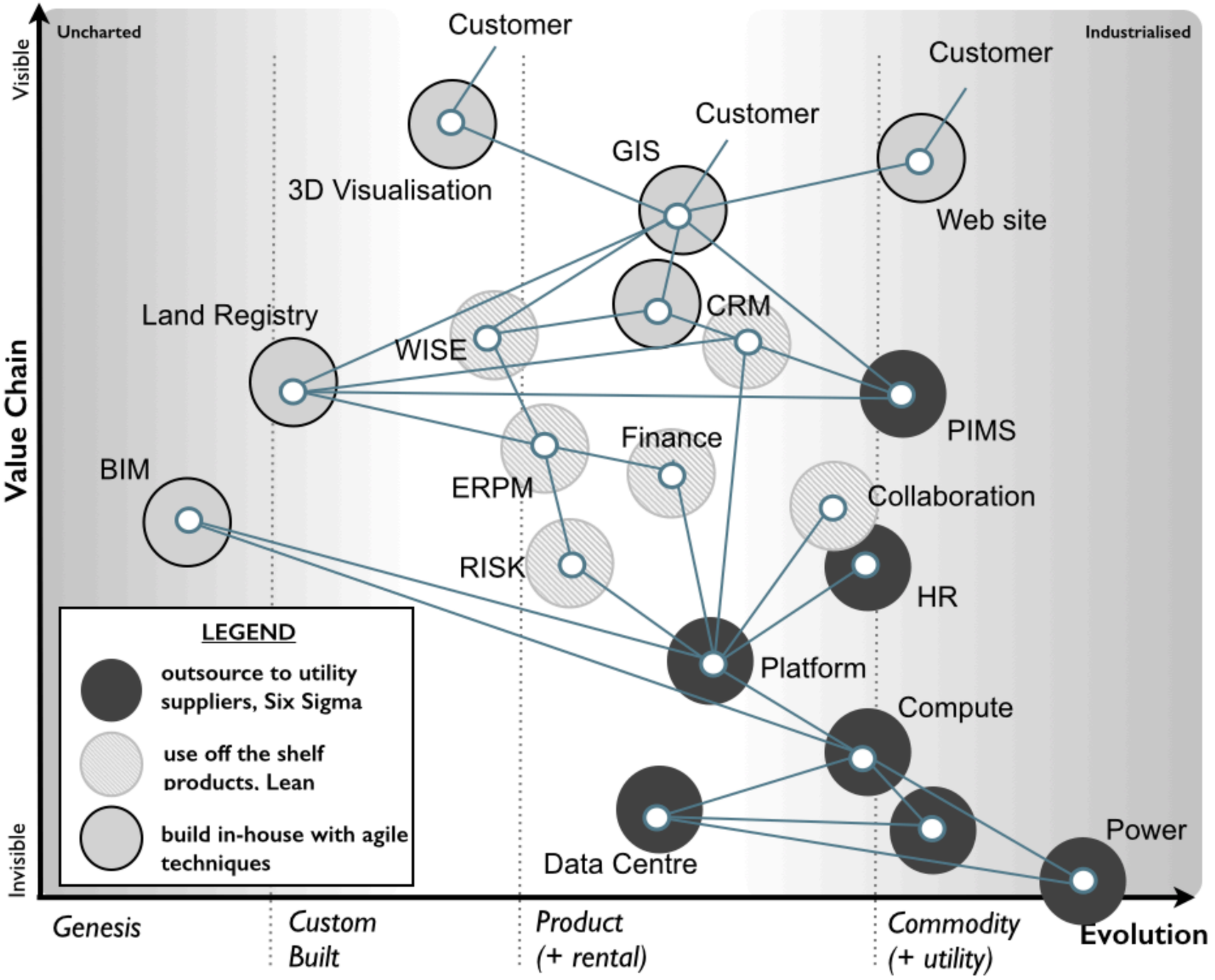


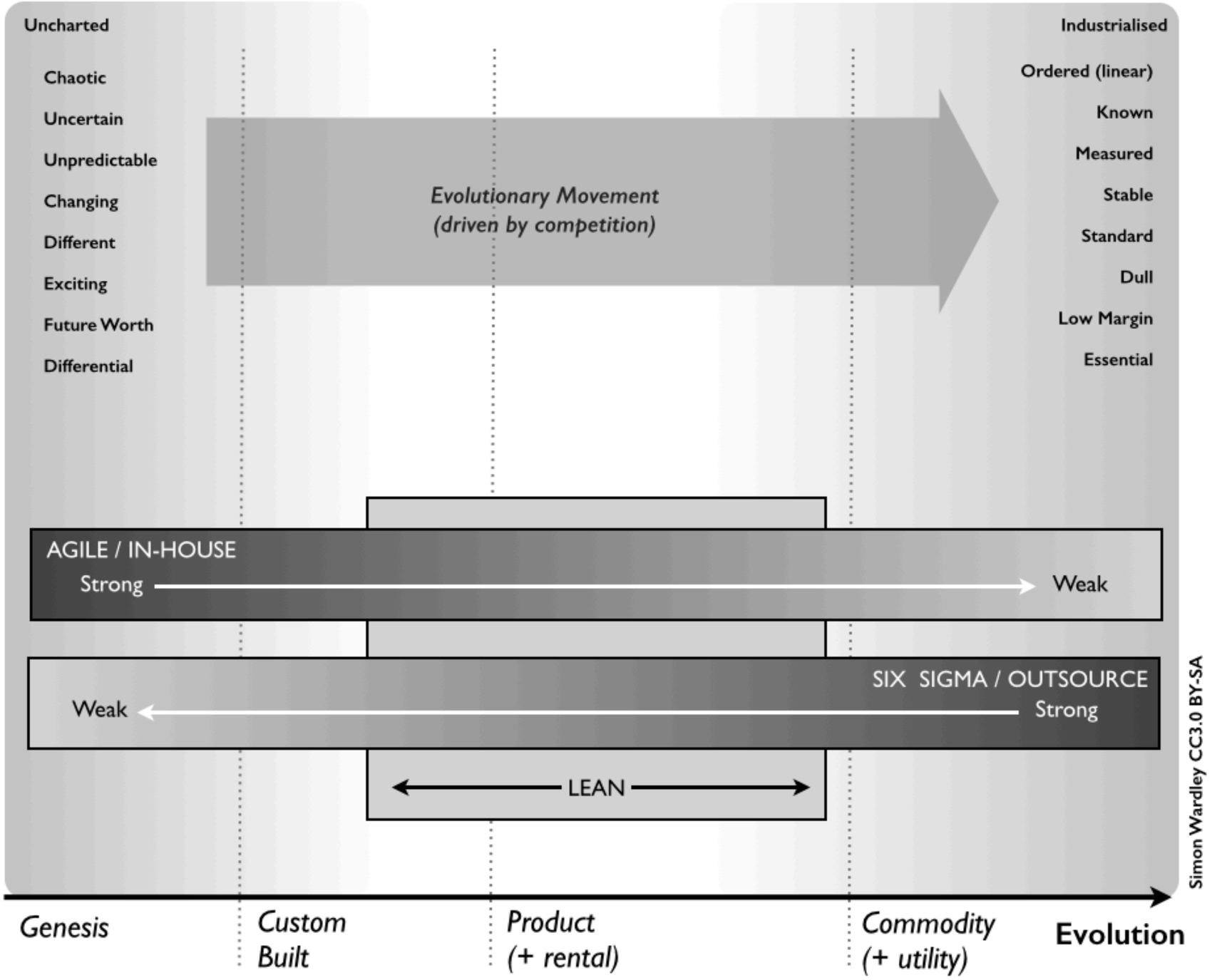
RECONCILED: PUNCTUATED GRADUALISM




STRANGLER FIG PATTERN




Uncharted
Industrialised
Visible
Invisible
Value Chain
Customer
Customer
Customer
3D Visualisation
GIS
Web site
CRM
Land Registry
WISE
PIMS
Finance
BIM
ERPM
Collaboration
RISK
HR
Platform
Compute
Power
Data Centre
LEGEND
outsource to utility suppliers, Six Sigma
use off the shelf products. Lean
build in-house with agile techniques
Genesis
Custom Built
Product (+ rental)
Commodity (+ utility)
Evolution


Uncharted
Chaotic
Uncertain
Unpredictable
Changing
Different
Exciting
Future Worth
Differential
Industrialised
Ordered (linear)
Known
Measured
Stable
Standard
Dull
Low Margin
Essential
Evolutionary Movement
(driven by competition)
AGILE / IN-HOUSE
Strong
Weak
SIX SIGMA / OUTSOURCE
Weak
Strong
LEAN
Genesis
Custom
Built
Product
(+ rental)
Commodity
(+ utility)
Evolution
Simon Wardley CC3.0 BY-SA

	Wardley's Doctrine (universally useful patterns that a user can apply regardless of context)					
	Communication	*Development*	*Operation*	*Learning*	*Leading*	*Structure*
IV				Listen to your ecosystem	Exploit the landscape	Design for constant evolution
					There is no core	No single culture
III			Optimise flow	*Bias towards the new*	Commit to the direction	Provide purpose, mastery & autonomy
			Do better with less		Be the owner	
					Inspire others	Seek the best
			Set exceptional standards		Embrace uncertainty	
					Be humble	
II		Focus on the outcome	Manage inertia	Bias towards action	Move fast	Think small teams
		Think fast, inexpensive, restrained and elegant				
		Use appropriate tools	Manage failure		Strategy is iterative	Distribute power and decision making
		Be pragmatic				
		Use standards	Effectiveness over efficiency			Think aptitude and attitude
	A bias towards open					
Phase I	Common Language	Know your users	Know the details	Bias towards data	*STEVE PURKIS VARIATION	
	Challenge Assumptions	Focus on user needs				
	Understand what is being considered	Remove bias and duplication				
		Use appropriate methods				

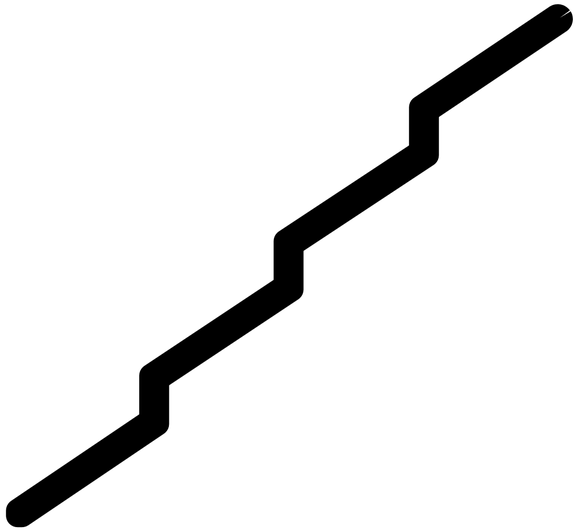
IT IS TIME TO STOP DOING SOFTWARE DEVELOPMENT AND IMPLEMENTATION “PROJECTS” - AND START SIMPLY DEVELOPING AND IMPLEMENTING SOFTWARE.

WE KNOW WE ARE GOING TO SPEND MONEY ON DEVELOPING AND IMPLEMENTING SOFTWARE. THAT IS A GIVEN.

WHAT WE NEED TO STOP DOING IS ARTIFICIALLY BREAKING SOFTWARE INTO “PROJECTS” WITH ALL THEIR OVERHEAD AND WASTE OF TIME AND MONEY.

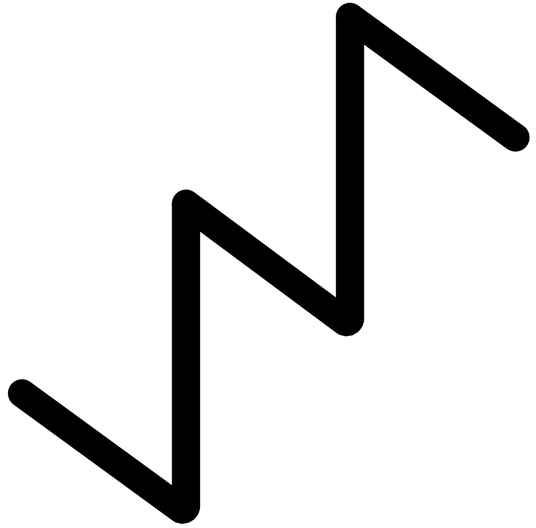
AS NIKE SAYS, “JUST DO IT.” START RUNNING - HALF A MILE TODAY, A MILE TOMORROW. THAT’S HOW YOU IMPROVE.

E



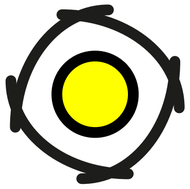
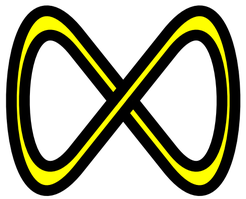
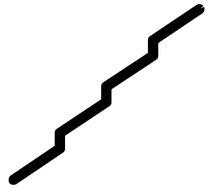
EVOLUTIONARY ENTERPRISE ARCHITECTURE

RATHER THAN

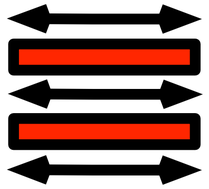
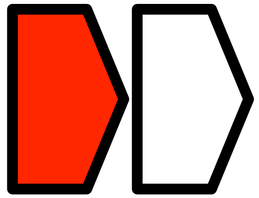
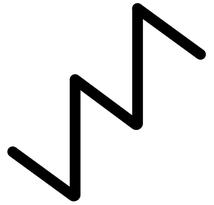


ENTROPY + BIG-BANG IMPROVEMENTS

Modern Enterprise Architecture

A	B	C	D	E
				
Aligning Value, People and Technology	Better Value Sooner Safer Happier	Continuous Conversational Governance	DevOps at Enterprise Scale	Evolutionary Enterprise Architecture

rather than…

				
Bilateral Alignments	De-duplication Standardisation Cost reduction	Phase-gated Document-led Adversarial Governance	Change not Run Dev not Ops Project not Product	Entropy + Big Bang Improvements

THANK YOU!

https://linkedin.com/in/simonrohrer

