NYU COURANT INSTITUTE OF MATHEMATICAL SCIENCES

DevOps: Culture, Agile Development, & Cloud Native Technologies

Fall 2020, CSCI-GA 2820, Graduate Division, Computer Science

Instructor:

John J Rofrano

Senior Technical Staff Member | DevOps Champion

IBM T.J. Watson Research Center

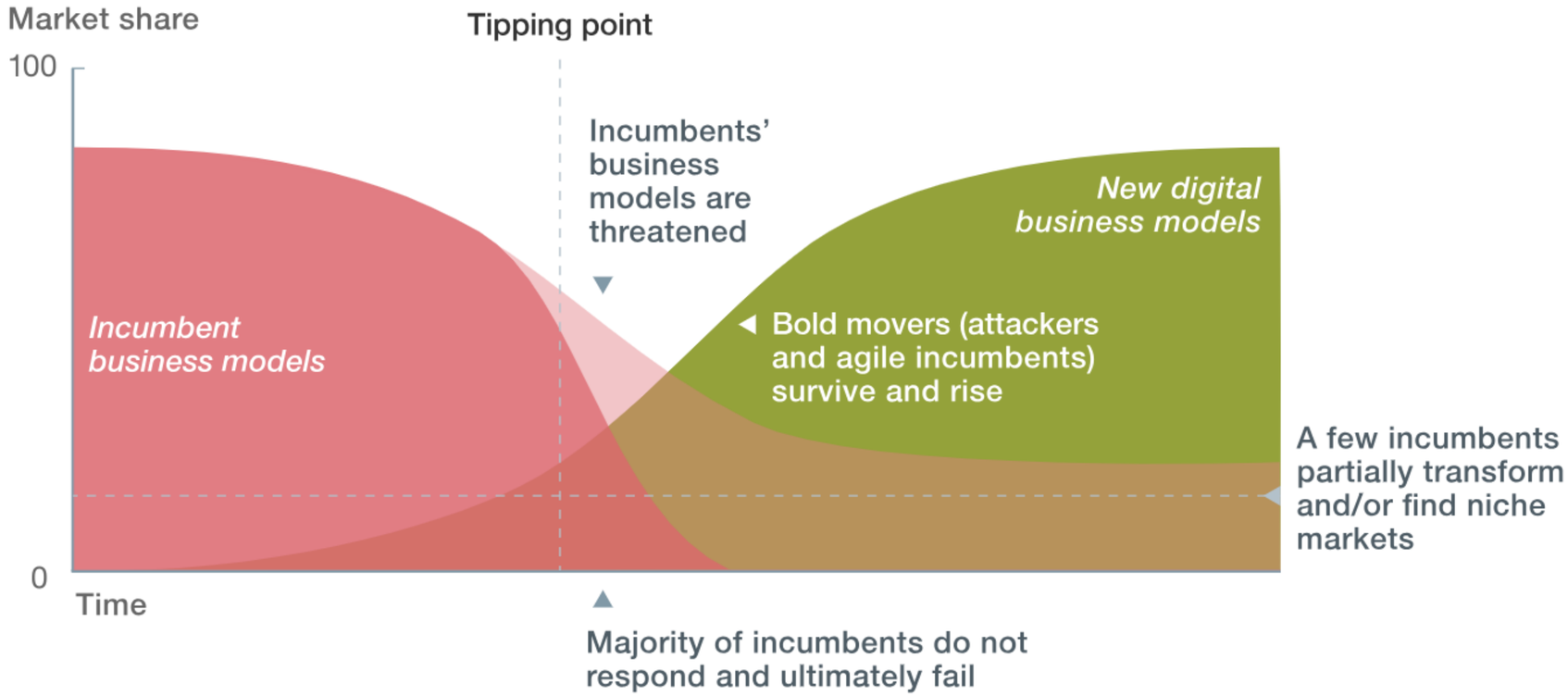
rofrano@cs.nyu.edu (@JohnRofrano)

What are businesses up against?

52%

of the Fortune 500 have disappeared since the year 2000

Disruptive Business Models



Example:
Call Taxi Service → Uber

8%

of companies believe their business model will remain economically viable through digitization

Source: McKinsey Digital Global Survey, 2016 and 2017; McKinsey analysis

Don’t Underestimate Impact of Digitization + Business Model

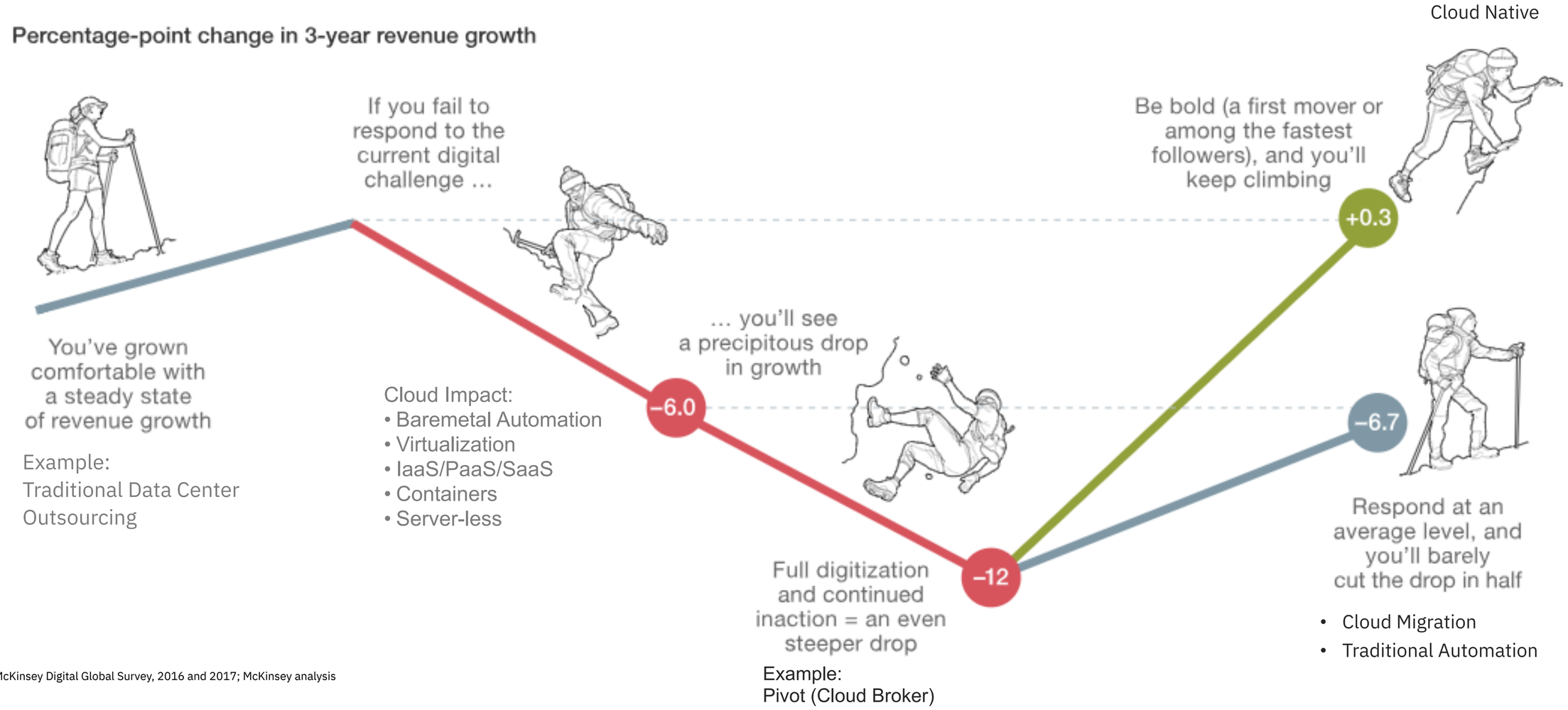
A ridesharing service is 40% cheaper than a regular cab for a 5 mile trip into Los Angeles

$$$ Ridesharing

$$$$$ Taxi

When was the last time you used a travel agent, bought a GPS device, Or carried a point-and-shoot camera separate from your phone?

Make bold moves or be irrelevant.



Source: McKinsey Digital Global Survey, 2016 and 2017; McKinsey analysis

“People don't buy drills… they buy holes”

Unlearn what you have learned

Often easier said than done

Consider this...

- **What if you could fail fast and roll back quickly?**
  - Then the impact of failure would be a minimal "blast radius"
- **What if you could test in-market instead of analyzing?**
  - Then you could experiment with customers instead of second-guessing them
- **What if your application design allowed individual components to be replaced?**
  - Then you wouldn't have "big bang" release weekends

Allspaw @ Velocity 2009

https://www.youtube.com/watch?v=LdOe18KhtT4

- Most people will cite John Allspaw (@allspaw) and Paul Hammond's Velocity 2009 presentation titled "**10+ Deploys Per Day: Dev and Ops Cooperation at Flickr**" as a pivotal moment in the Devops movement.

- "In the last week there were **67** deploys of **496** changes by **18** people" – Flickr Dev Blog, **December 17th 2008**.

https://conferences.oreilly.com/velocity/velocity2009/public/schedule/detail/7641

2011 Etsy Deploys to Production every ~25 min*

- In January 2011 (a month in which we did over **a billion** page views)
- Code committed by **76** unique individuals
- Was deployed to production by **63** different folks
- A total of **517** times

* Based on 22 days 10 hrs/day (517 / 22 / 10 = 2.35/hr)

“Our deployment environment requires a lot of trust, transparency, communication, coordination, and discipline across the team.”

–Chad Dickerson, CTO Etsy

...but these are "mythical" companies, right?

...big enterprisers can't possibly work like this, right?

2016 DevOps Enterprise Summit

- **Ticketmaster** - 98% reduction in MTTR
- **Nordstrom** - 20% shorter Lead Time
- **Target** - Full Stack Deploy 3 months to minutes
- **USAA** - Release from 28 days to 7 days
- **ING** - 500 application teams doing DevOps
- **CSG** - From 200 incidents per release to 18

ticketmaster

NORDSTROM

USAA®

TARGET

ING

How are they doing this?

They have embraced the DevOps culture

What is DevOps?

“The term (development and operations) is an extension of agile development environments that aims to enhance the process of software delivery as a whole.”

–Patrick Debois, 2009

What is DevOps?

DevOps is a recognition that Development and Operations needs to stop working alone in their "siloed" towers and start working together

- To do this we need:
  - A **culture** of collaboration valuing openness, trust, and transparency
  - An **application design** that does not require entire systems to be redeployed just to add a single function
  - **Automation** that accelerates and improves the consistency of application delivery so that we can develop and deliver software with speed and stability
  - A dynamic software-defined, **programmable platform** to continuously deploy onto

“DevOps is the practice of development and operations teams working together in the entire software lifecycle, following lean and agile principles that allow them to deliver software in a rapid and continuous manner.”

Things DevOps is Not

- DevOps is **not** simply combining Development & Operations teams
- DevOps is **not** a separate team
- DevOps is **not** a tool
- DevOps is **not** a one-size-fits-all strategy
- DevOps is **not** just automation

What's so hard about getting Dev and Ops to work together?

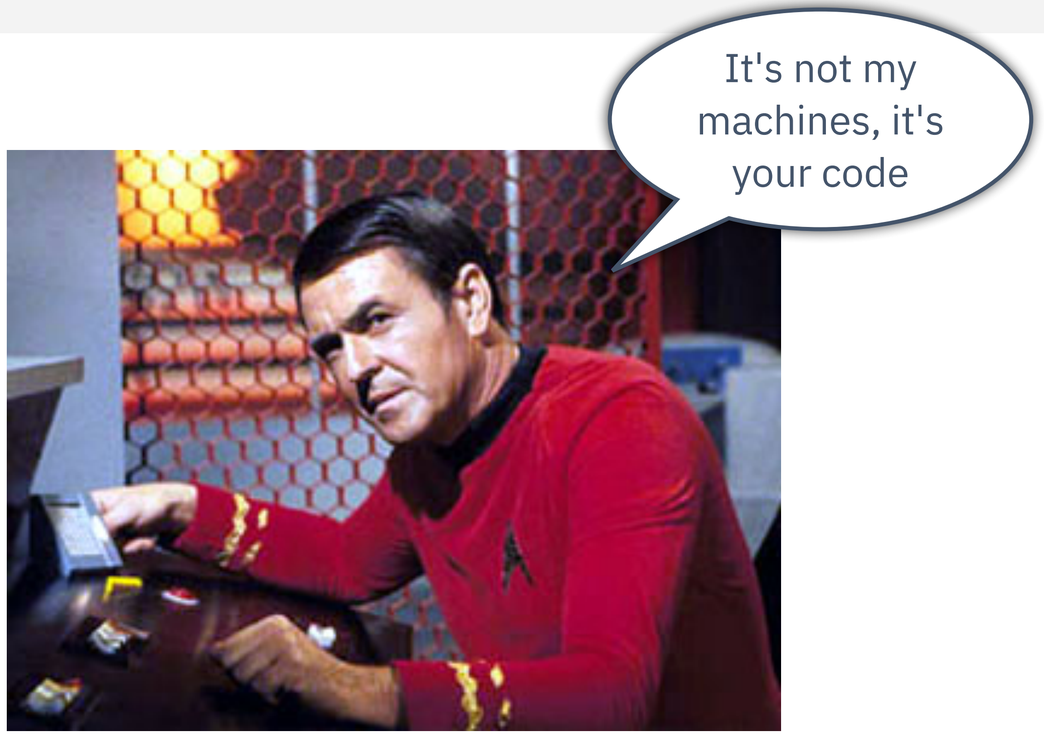
Dev verses Ops

Little bit weird

Sits closer to the boss

Thinks too hard

vs



Pulls levers & turns knobs

Easily excited

Yells a lot in emergencies

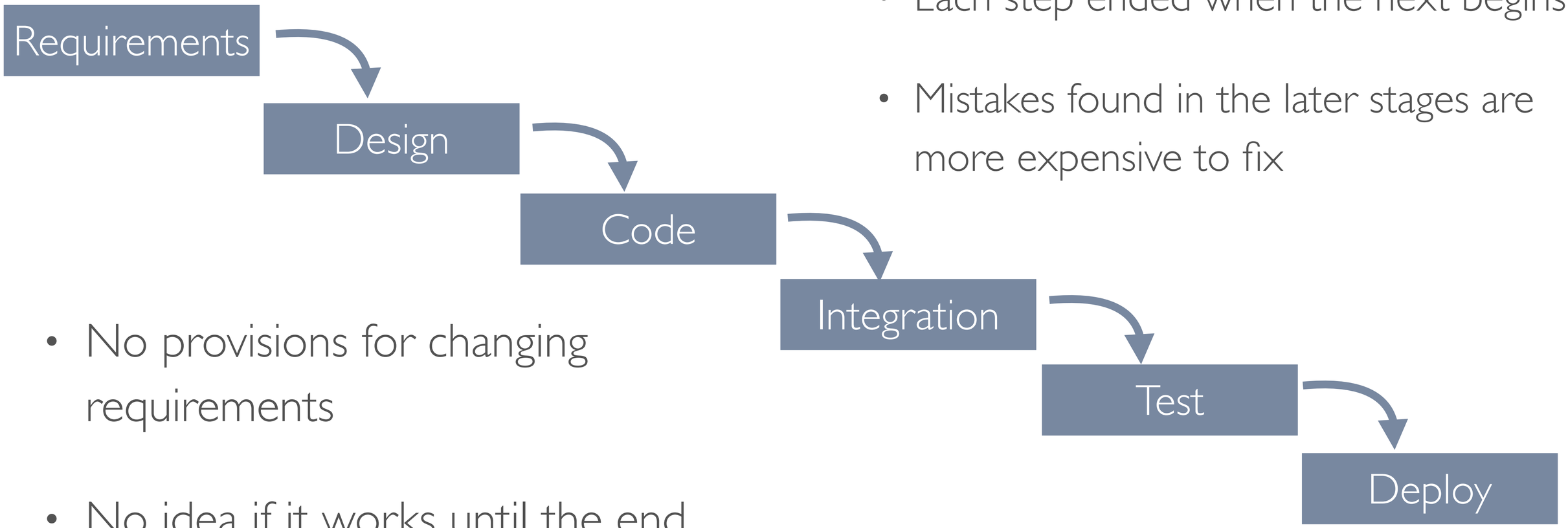
To fully appreciate where we are... We must first understand from where we came

BRIEF HISTORY OF DEVOPS

Pre-DevOps History Review

- Architects worked for months designing the system on paper and then released the documents to development
- Development worked for months on features in an isolated development environment and then released the code to testing
- Testing opened defects and sent the code back to development until no more sev 1 or 2’s
- At some point, development then released the code to operations for deployment
- The operations team took forever to deploy the application and then kept it running from that point on

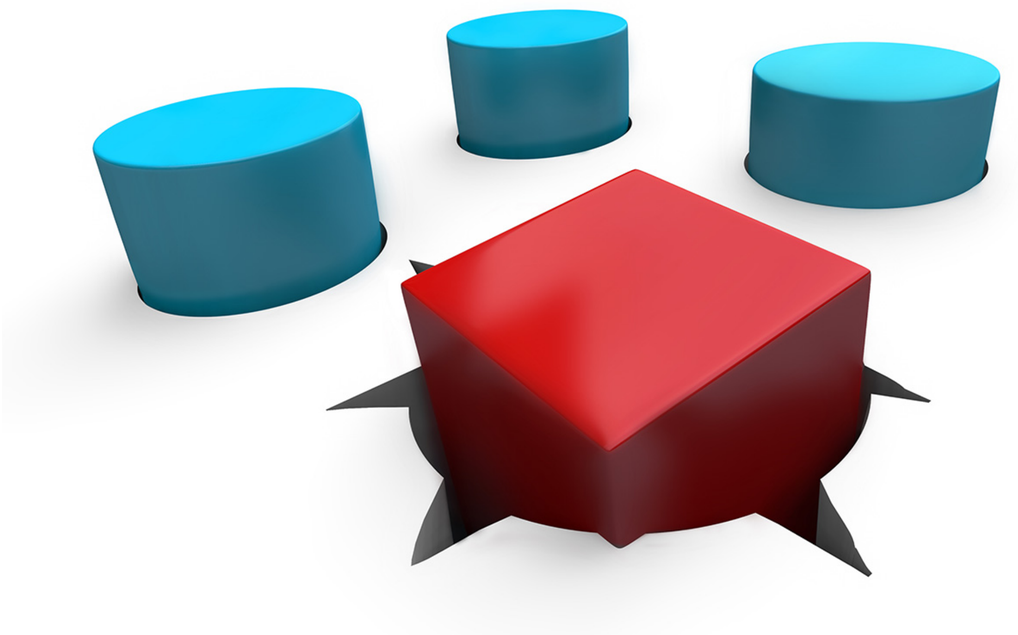
Traditional Waterfall Development



- Each step ended when the next begins
- Mistakes found in the later stages are more expensive to fix
- No provisions for changing requirements
- No idea if it works until the end

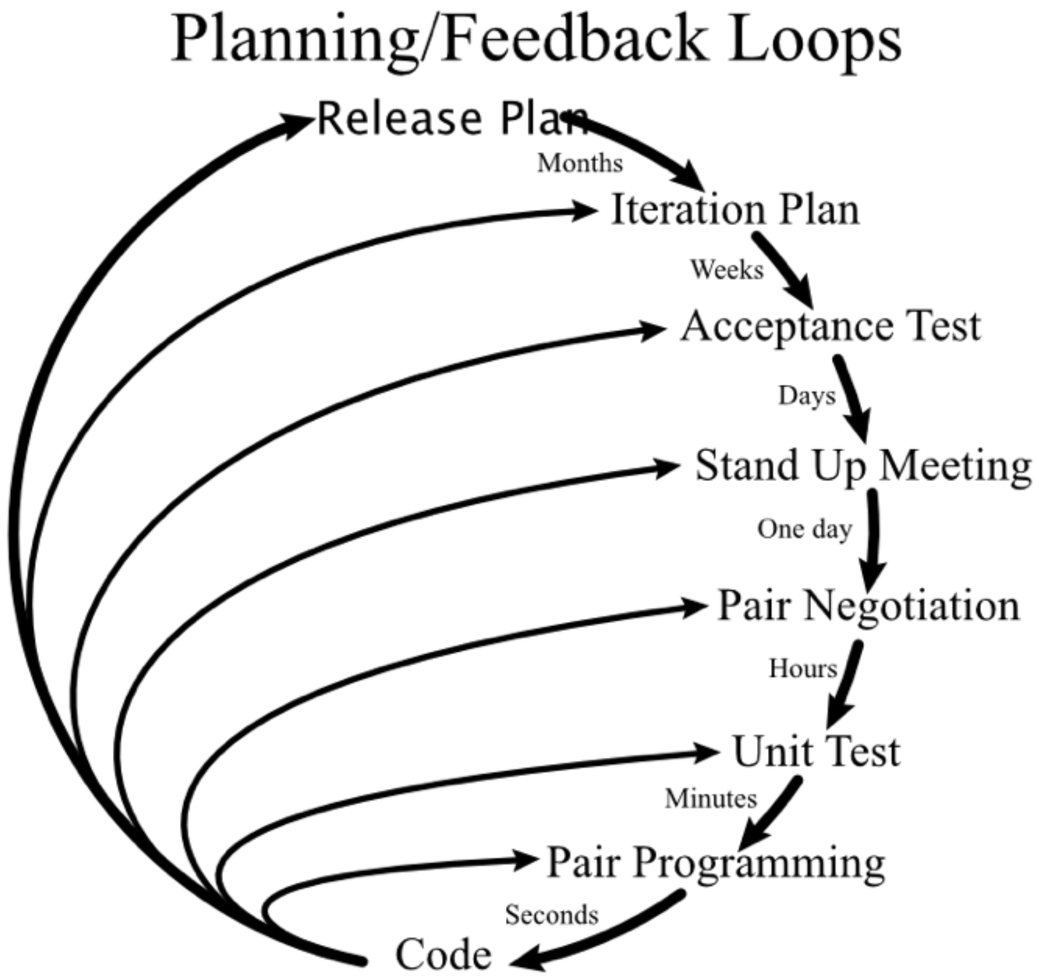
Problems with this approach?

- There was usually a long time between software releases
- Because all of the teams worked separately, the development team was not always aware of operational roadblocks that might prevent the program from working as anticipated
- The people the furthest from the code who knew the least about it were deploying it into production



Extreme Programming (XP)

- In 1996 Kent Beck introduced Extreme Programming
- Based on an interactive approach to software development
- Intended to improve software quality and responsiveness to changing customer requirements
- Was one of the first agile methods

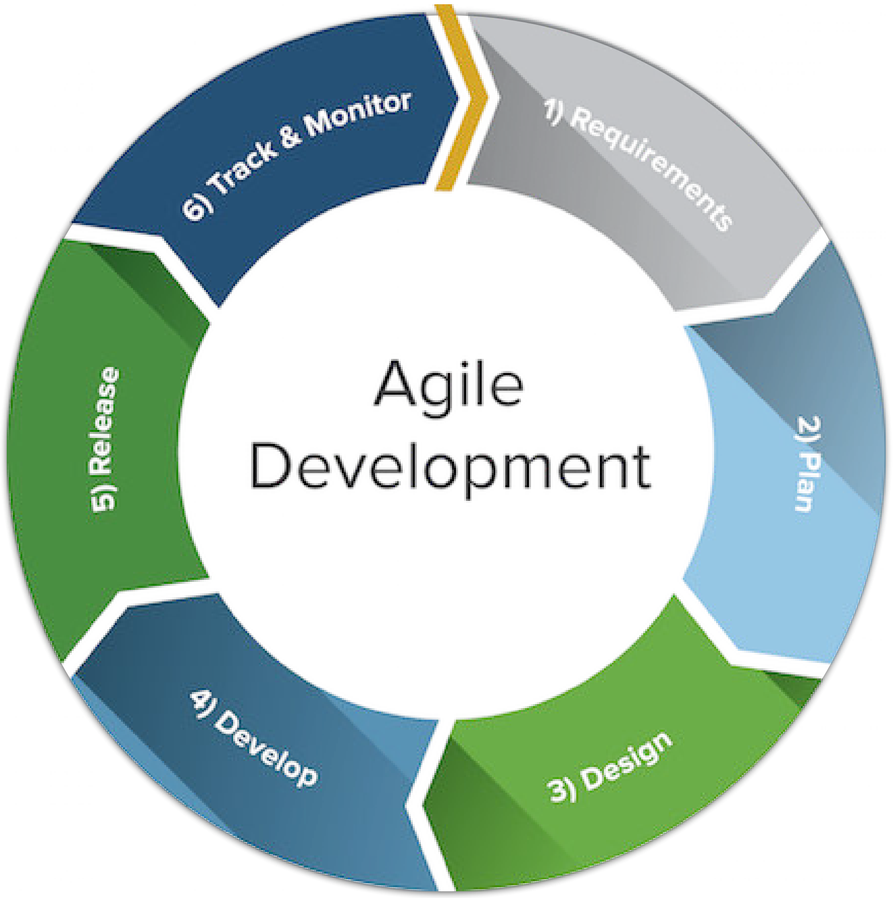


Agile Manifesto

- In 2001, seventeen software developers met at a resort in Snowbird, Utah to discuss these lightweight development methods
- Including among others Kent Beck, Ward Cunningham, Dave Thomas, Jeff Sutherland, Ken Schwaber, Jim Highsmith, Alistair Cockburn, and Bob Martin
- Together they published the Manifesto for Agile Software Development

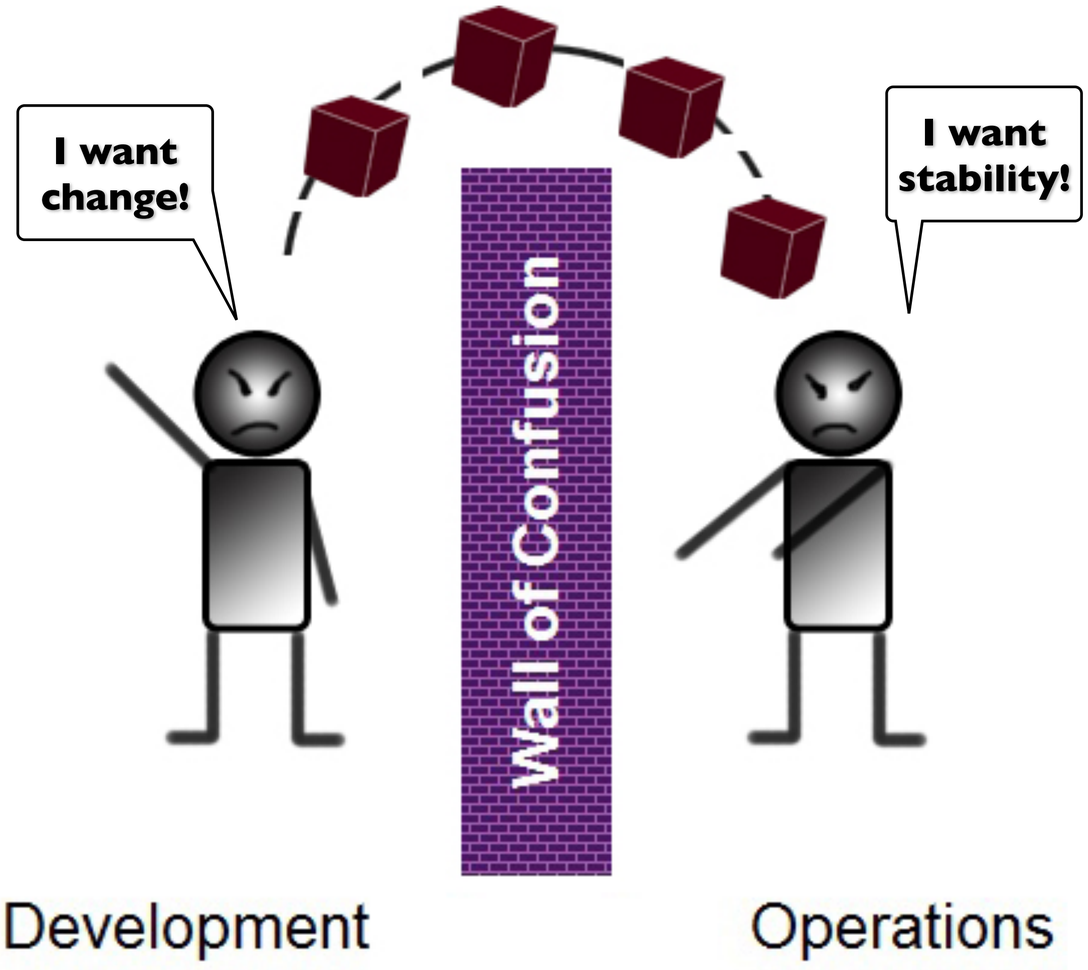
Agile Development

- Requirements and solutions evolve through the collaborative effort of **self-organizing** and **cross-functional** teams and their customers
- It advocates **adaptive planning**, evolutionary development, early delivery, and **continual improvement**
- It encourages rapid and flexible **response to change**



The Agile Dilemma

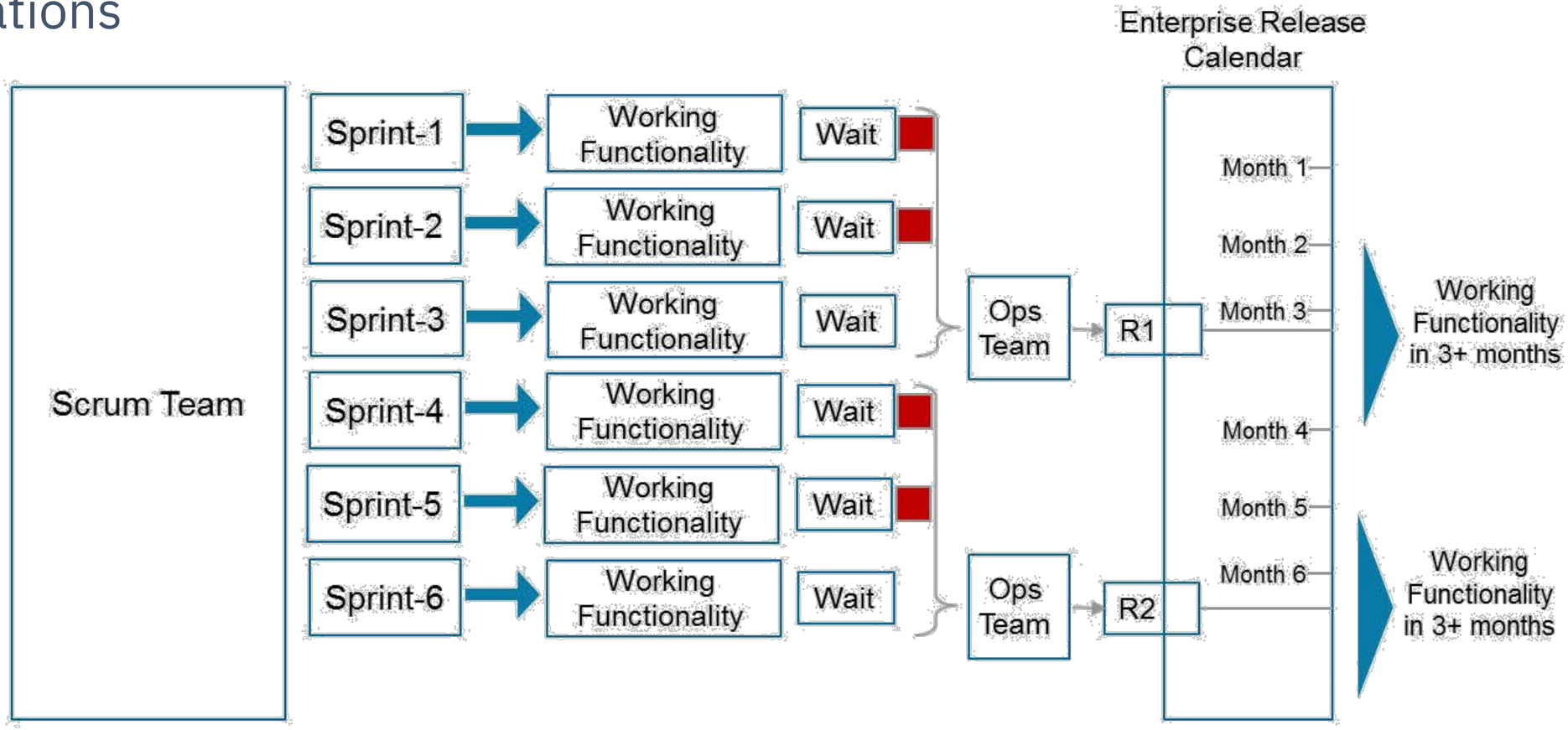
- While Agile improved the speed and accuracy of software for developers
- It did nothing for operations
- Many development teams just got frustrated by ops not being able to deliver at the speed of development



Source: Andrew Clay Shafer, Senior Director of Technology at Pivotal

Why isn’t Agile alone good enough?

- While Agile improved the speed and accuracy of software for developers, it did nothing for operations

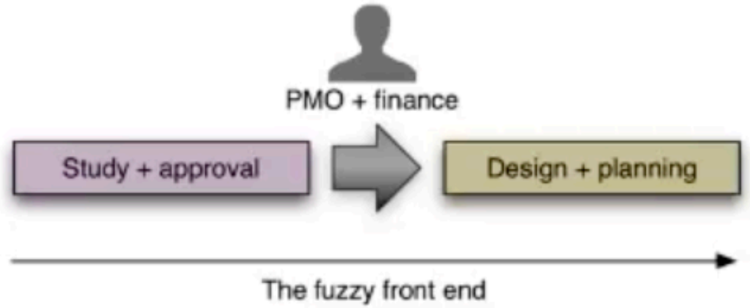


https://www.scrumalliance.org/community/articles/2014/april/devops-and-agile

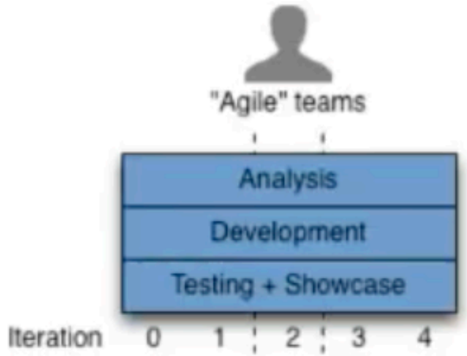
Dysfunctional DevOps

a.k.a Water-Scrum-Fall

Organizations that wants to change without actually changing anything!



water

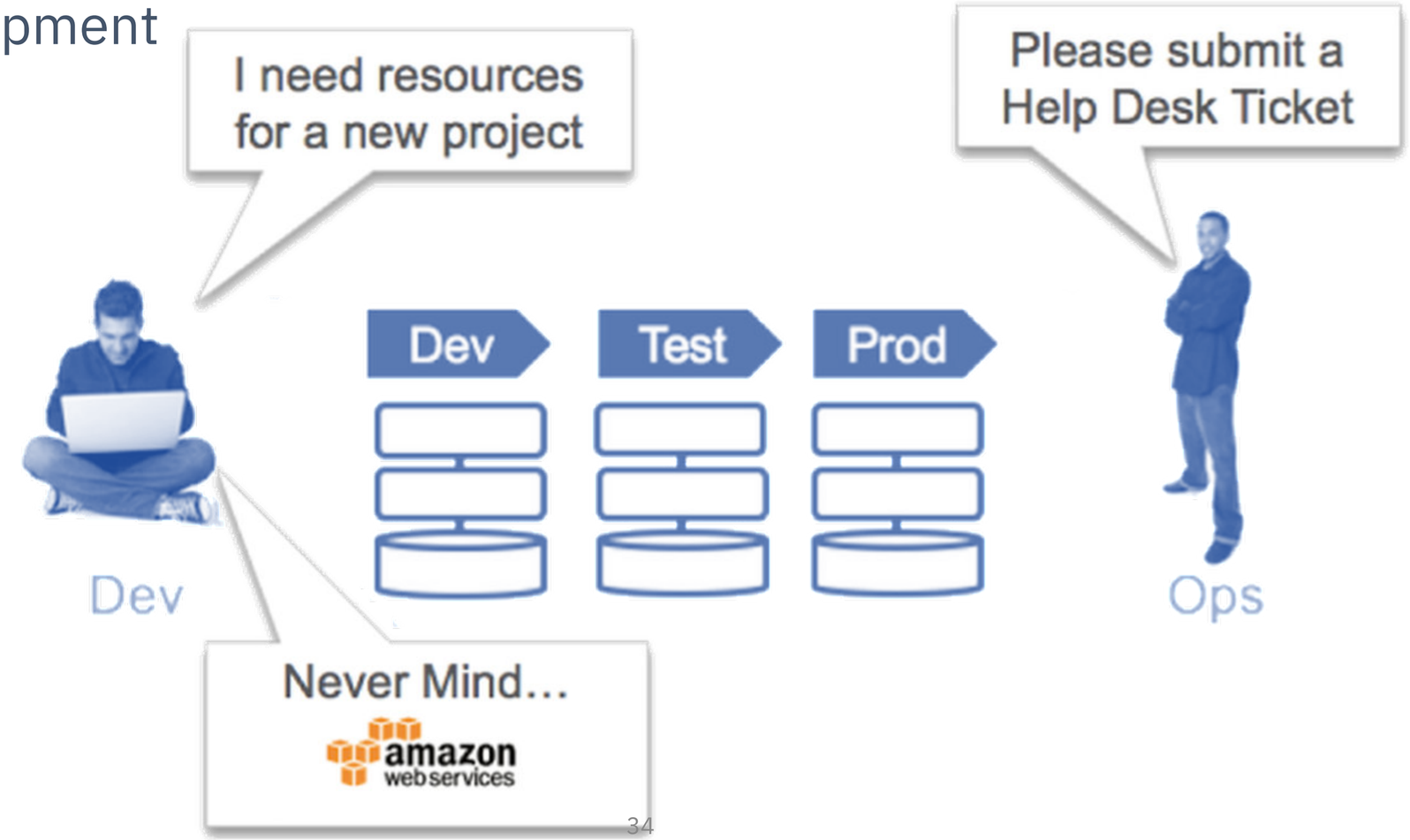


scrum

fall

The results of 2 speed IT

- Many development teams just got frustrated by ops not being able to deliver at the speed of development



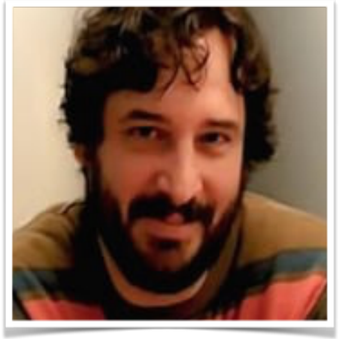
This is how "Shadow IT" is born

...enter DevOps

DevOps Started in 2007

2007 Patrick Debois

- Recognized Dev and Ops worked ineffectively and not together



2008 Andrew Clay Shafer

- Agile 2008 Conference BoF "Agile Infrastructure"

2009 John Allspaw

- Velocity 2009 “10+ Deploys Per Day: Dev and Ops Cooperation at Flickr”

DevOpsDays 2009

- Patrick Debois (often called the Father of DevOps) created the first DevOpsDays conference in Ghent, Belgium in October 2009
- It was "**The conference that brings development and operations together**"
  - This is where the term DevOps was first used
- DevOpsDays is now a local conference held internationally several times a year in different cities

https://www.devopsdays.org

DevOpsDays 2019

40 Events in
21 countries
are scheduled
for 2019
(10 years later)

FEB 7 - 8, 2019
Charlotte

FEB 21 - 22, 2019
Geneva

MAR 8, 2019
Los Angeles

MAR 9, 2019
Natal

MAR 29 - 30, 2019
Vancouver

APR 3 - 4, 2019
Copenhagen

APR 9 - 10, 2019
Tokyo

APR 9 - 10, 2019
Atlanta

APR 10 - 11, 2019
Jakarta

APR 10 - 11, 2019
São Paulo

APR 16 - 17, 2019
Houston

APR 23 - 24, 2019
Seattle

APR 24 - 25, 2019
Baltimore

APR 29 - 30, 2019
Denver

MAY 2 - 3, 2019
Austin

MAY 2 - 3, 2019
Des Moines

MAY 9 - 10, 2019
Nashville

MAY 14 - 15, 2019
Zürich

MAY 14 - 15, 2019
Salt Lake City

MAY 17 - 18, 2019
Kyiv

MAY 24 - 25, 2019
Porto Alegre

MAY 29 - 30, 2019
Toronto

MAY 30, 2019
Boise

JUN 3 - 4, 2019
Portugal

JUN 8, 2019
Aracaju

JUN 25 - 28, 2019
Amsterdam

JUL 25 - 26, 2019
Indianapolis

AUG 6 - 7, 2019
Minneapolis

SEP 9, 2019
Cairo

SEP 10 - 12, 2019
Portland

SEP 19 - 20, 2019
Istanbul

SEP 23 - 24, 2019
Boston

SEP 26 - 27, 2019
London

OCT 1 - 2, 2019
Raleigh

OCT 16 - 18, 2019
Taipei

OCT 22 - 23, 2019
Oslo

OCT 22 - 23, 2019
Philadelphia

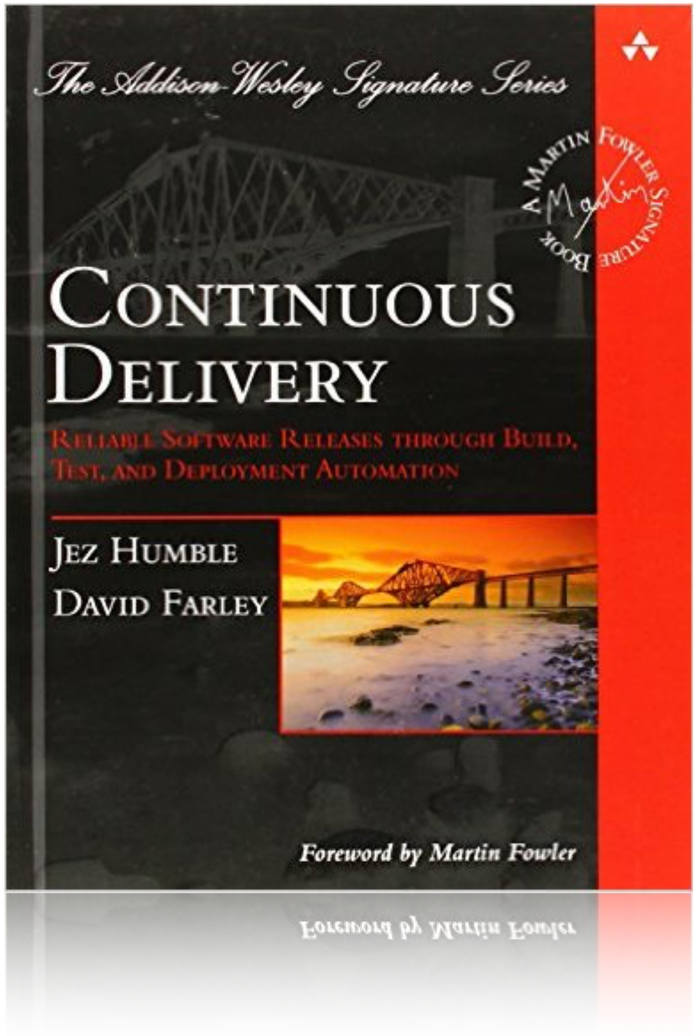
OCT 23 - 24, 2019
Detroit

OCT 28 - 30, 2019
Ghent

NOV 27 - 28, 2019
Berlin

2011 Continuous Delivery

- Jez Humble and David Farley write this groundbreaking book that sets out the principles and technical practices that enable **rapid, incremental delivery of high quality**, valuable new functionality to users
- Through automation of the build, deployment, and testing process, and improved collaboration between developers, testers, and operations, delivery teams can get **changes released in a matter of hours —sometimes even minutes–** no matter what the size of a project or the complexity of its code base

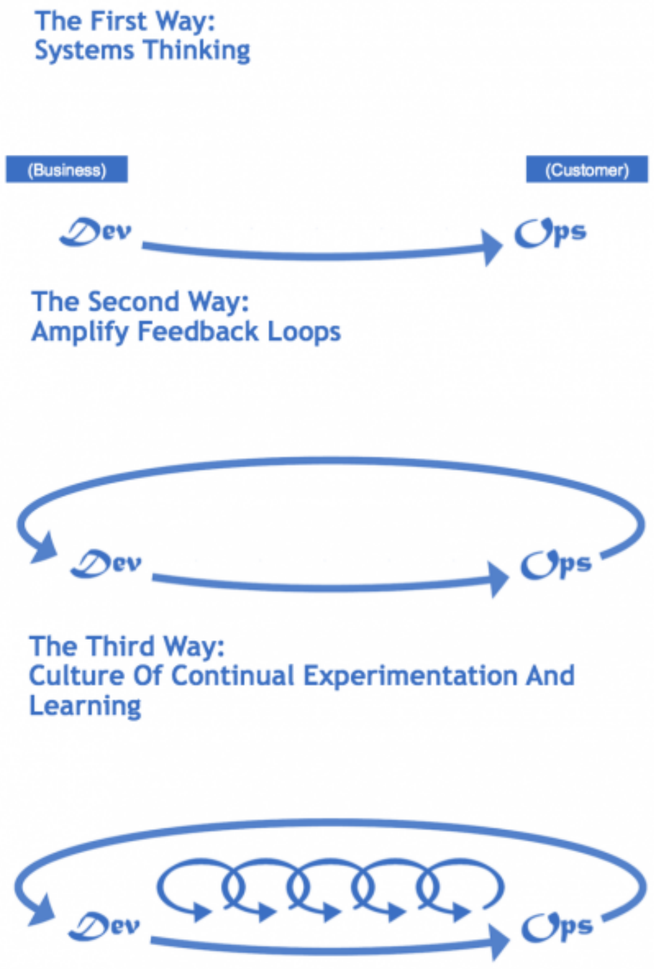


2015 The Phoenix Project

- Gene Kim, Kevin Behr, and George Spafford write this seminal book on running Lean Operations
- **Based on The Goal**: A Process of Ongoing Improvement by Eliyahu M. Goldratt
- Updated from a **Lean Manufacturing** story to Software Development and Delivery

The Three Ways

The Phoenix Project - Gene Kim



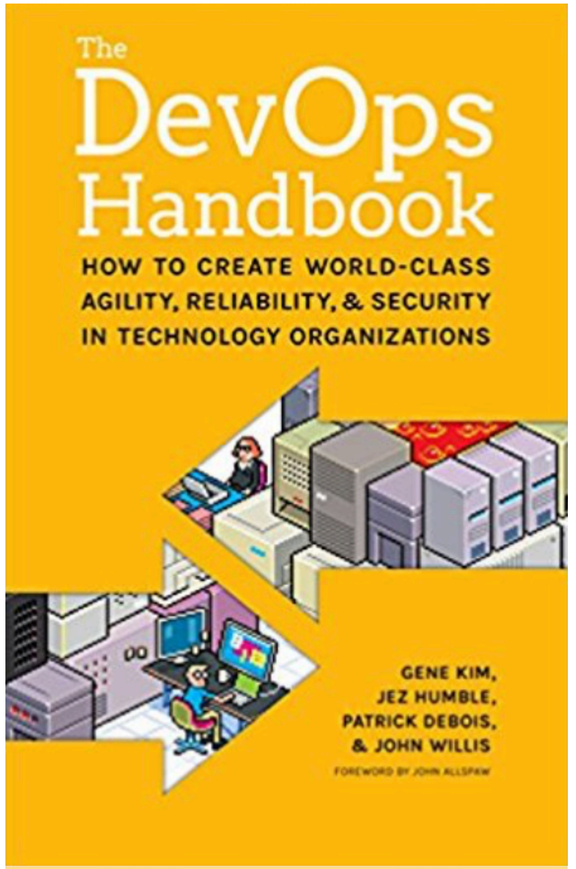
The First Way helps us understand how to create fast flow of work as it moves from Development into IT Operations, because that's what's between the business and the customer.

The Second Way shows us how to shorten and amplify feedback loops, so we can fix quality at the source and avoid rework.

And **the Third Way** shows us how to create a culture that simultaneously fosters experimentation, learning from failure, and understanding that repetition and practice are the prerequisites to mastery.

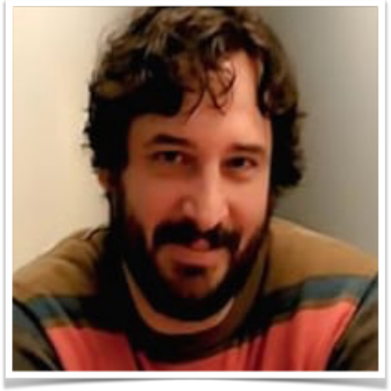
2016 The DevOps Handbook

- Gene Kim, Jez Humble, Patrick Debois, and John Willis write this follow-on to The Phoenix Project
- Serves as a practical guide on how to implement the concepts introduced in The Phoenix Project
- John Willis worked at Docker and Chef and is a DevOpsDays coordinator after being at the original in Ghent 2009



Major Influencers of the early DevOps Movement

Patrick Debois



Andrew Clay Shafer

John Allspaw

Jez Humble

Gene Kim

John Willis

Why is the history important?

It reminds us that DevOps is...

- from the practitioners, by practitioners
- not a product, specification, job title
- an experience-based movement
- decentralized and open to all

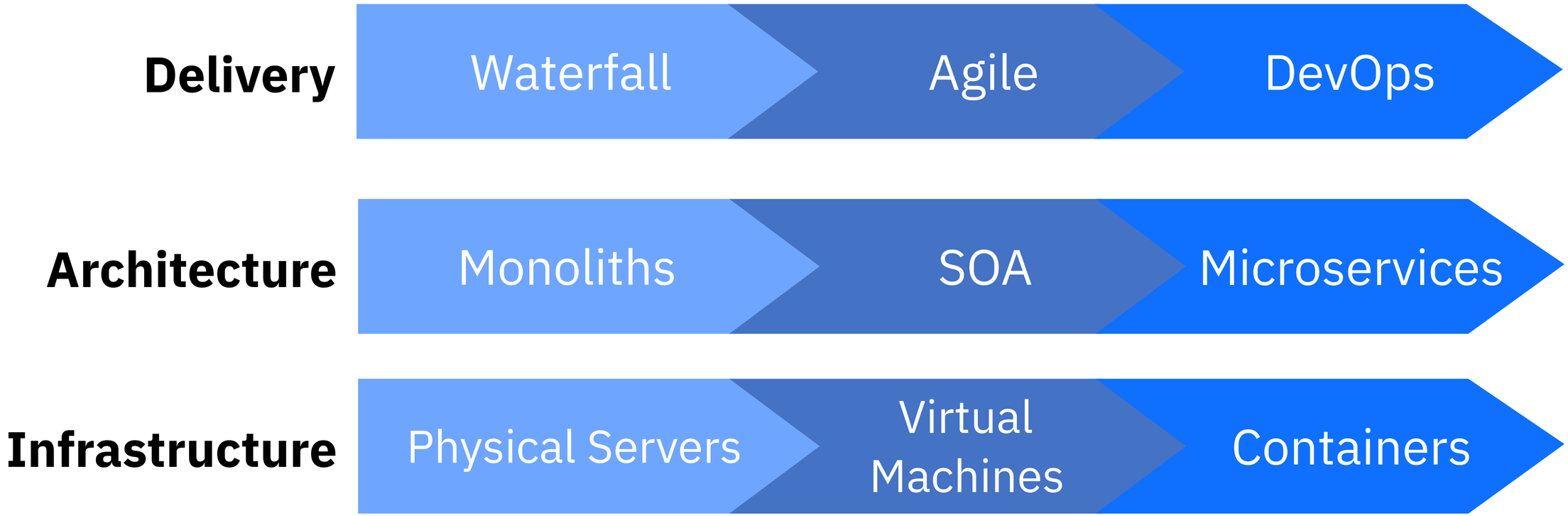
Damon Edwards (https://www.youtube.com/watch?v=o7-IuYS0iSE)

What is the Goal?

Agility is the goal

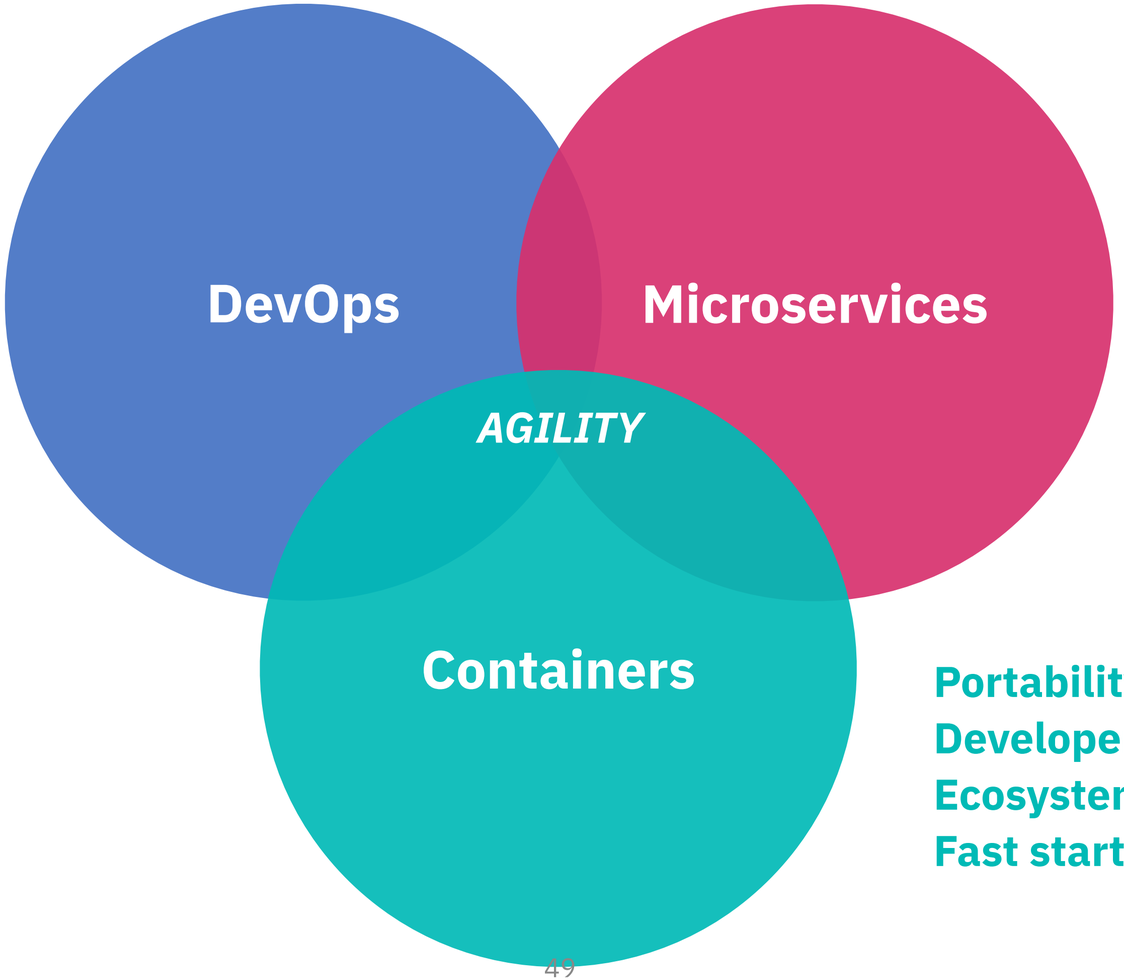
- Smart experimentation
- Moving in–market with maximum velocity and minimum risk
- Gaining quick valuable insight to continuously change the value proposition and quality

Application Evolution



Agility: The Three Pillars

Cultural Change
Automated Pipeline
Everything as Code
Immutable Infrastructure



Loose Coupling/Binding
RESTful APIs
Designed to resist failures
Test by break / fail fast

Portability
Developer Centric
Ecosystem enabler
Fast startup

The Perfect Storm

DevOps for speed and agility
Microservices for small deployments
Containers for ephemeral runtimes

"DevOps starts with learning how to work differently. It embraces cross-functional teams with openness, transparency, and respect as pillars."

–Tony Stafford, Shadow Soft

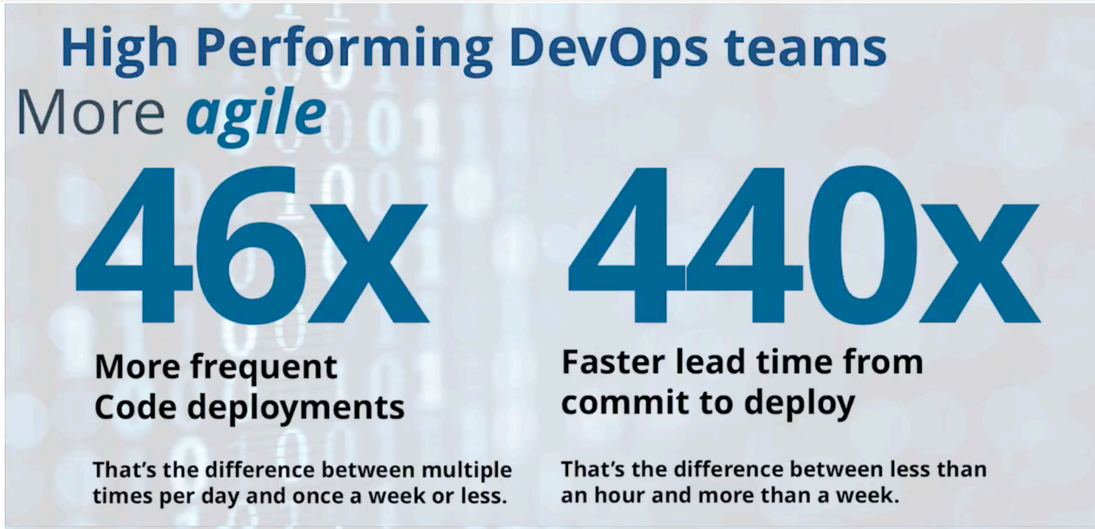
State of DevOps Report

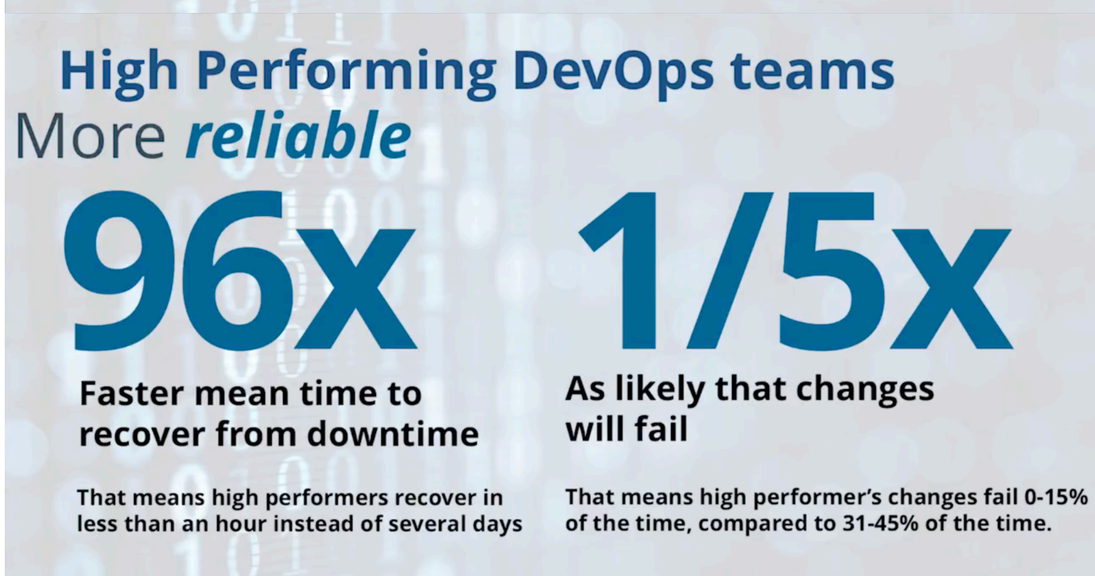
- Puppet Labs and DORA began measuring DevOps success in 2015 lead by Dr. Nicole Forsgren
- We have seen steady increase in companies adopting DevOps since then

Current DevOps Impact

State of DevOps Report by Puppet Labs 2017

- Taking an experimental approach to product development can improve your IT and organizational performance
- High-performing organizations are decisively outperforming their lower-performing peers in terms of throughput
- Undertaking a technology transformation initiative can produce sizeable cost savings for any organization





"Culture is the #1 success factor in DevOps. Building a culture of shared responsibility, transparency and faster feedback is the foundation of every high performing DevOps team."

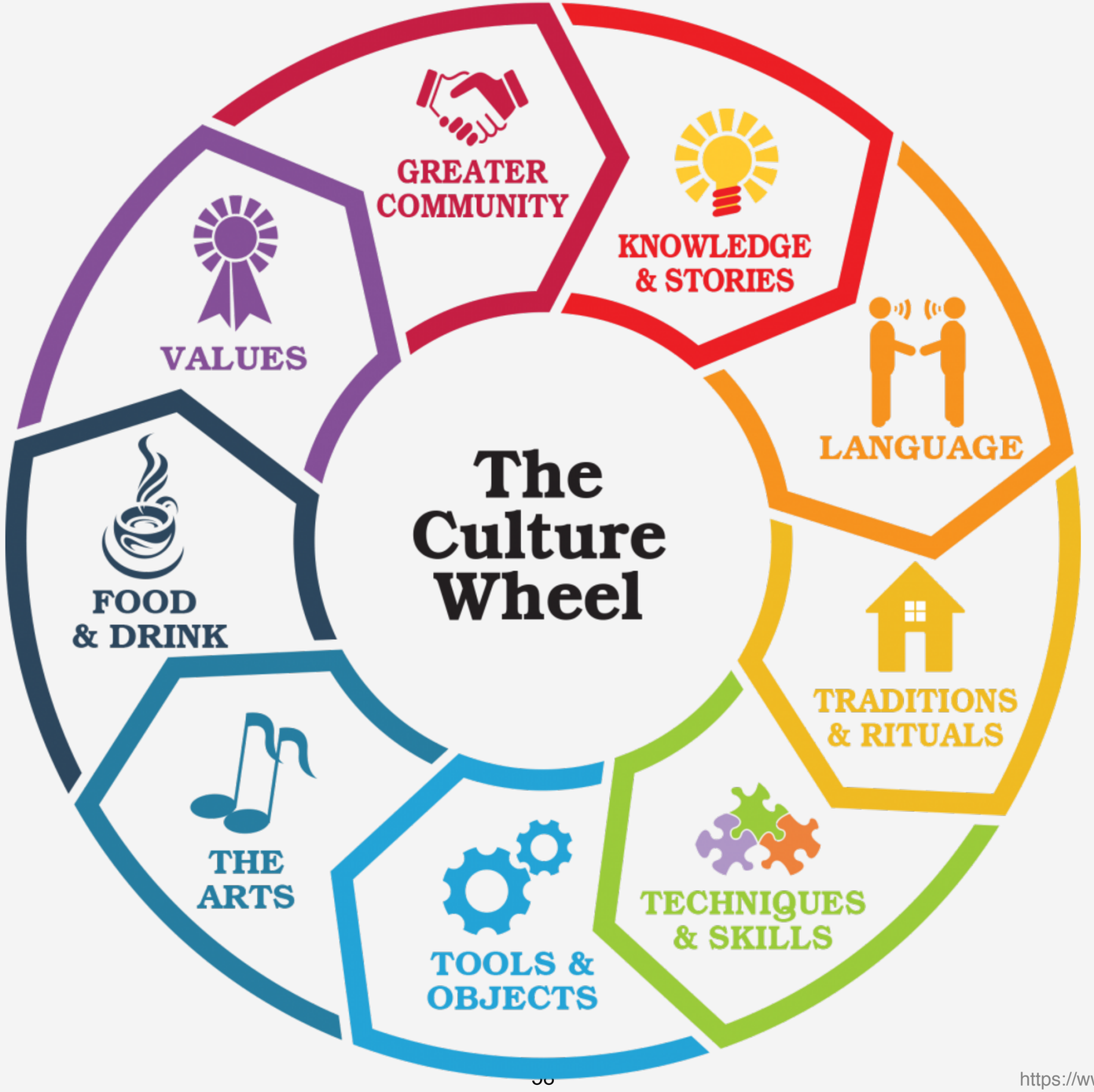
–Atlassian

DevOps has Three Dimensions

1. Culture
2. Methods
3. Tools

“While tools and methods are important
… it’s the culture that has the biggest impact”

How do you change a culture?


The Culture Wheel
GREATER COMMUNITY
KNOWLEDGE & STORIES
LANGUAGE
TRADITIONS & RITUALS
TECHNIQUES & SKILLS
TOOLS & OBJECTS
THE ARTS
FOOD & DRINK
VALUES

How do you change a culture?

- You must change the way people **think**
- You must change the way people **work**
- You must change the way people are **organized**
- You must change the way people are **measured**

You must change the way people think

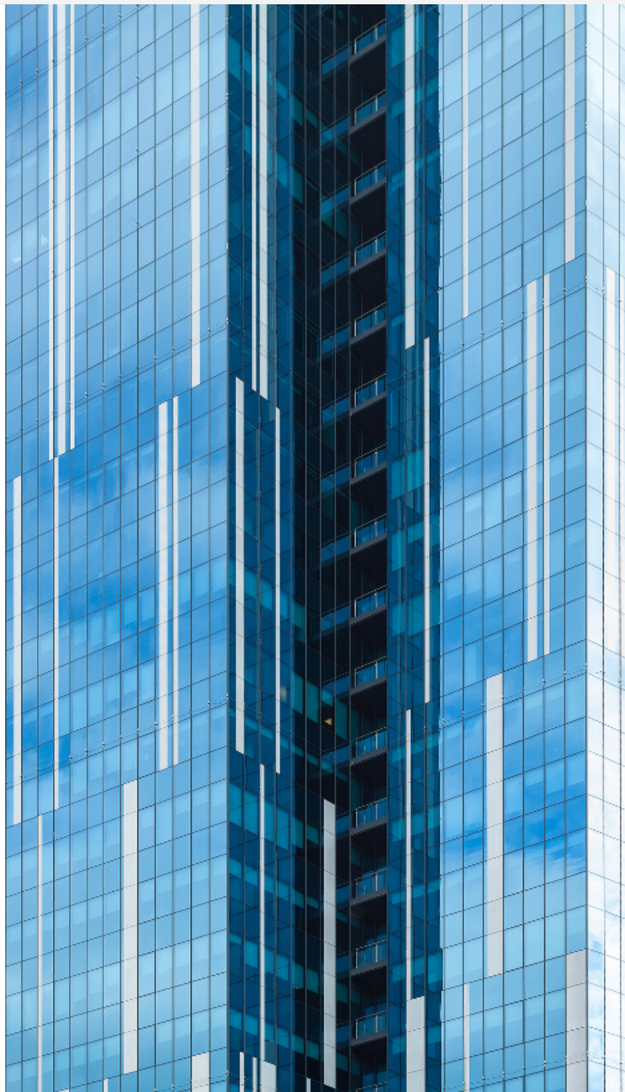
DevOps Thinking

- Social Coding
- Behavior and Test Driven Development
- Working in small batches*
- Build Minimum Viable Products for gaining insights*
- Failure leads to understanding

* from Lean Manufacturing

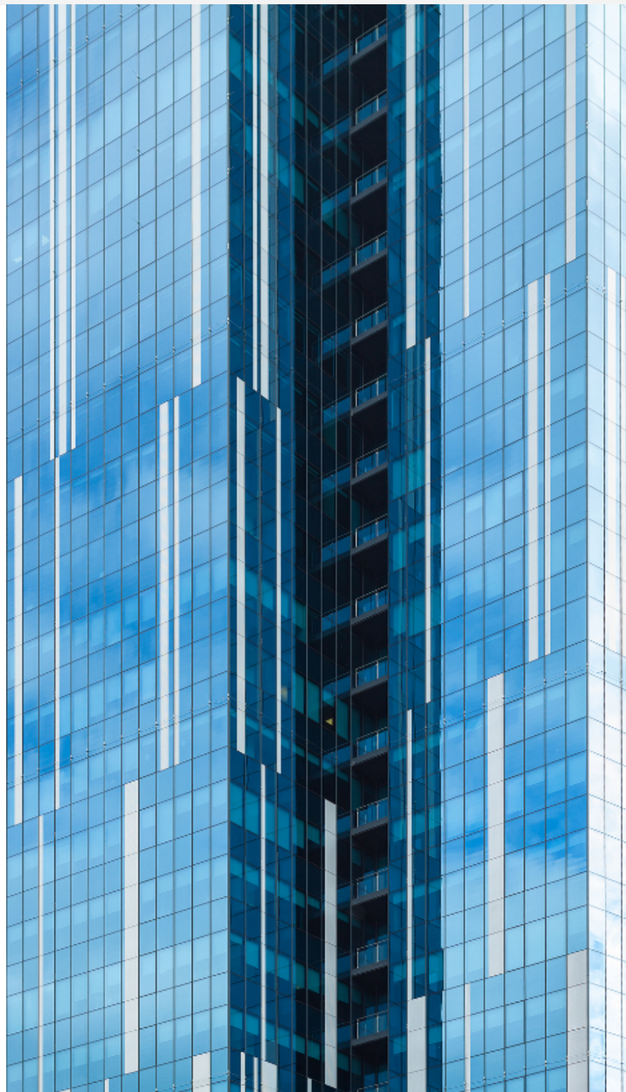
Traditional Thinking

- In the past developers worked on private repositories and you had to be a member of the team to contribute
- You see a project that is 80% of what you need but there are some missing features but...
- You feel that if you make a feature request of the project owner, your request will be rejected or go to the bottom of their priorities
- So you rebuild 100% of what you need so as not to have a dependency on another project



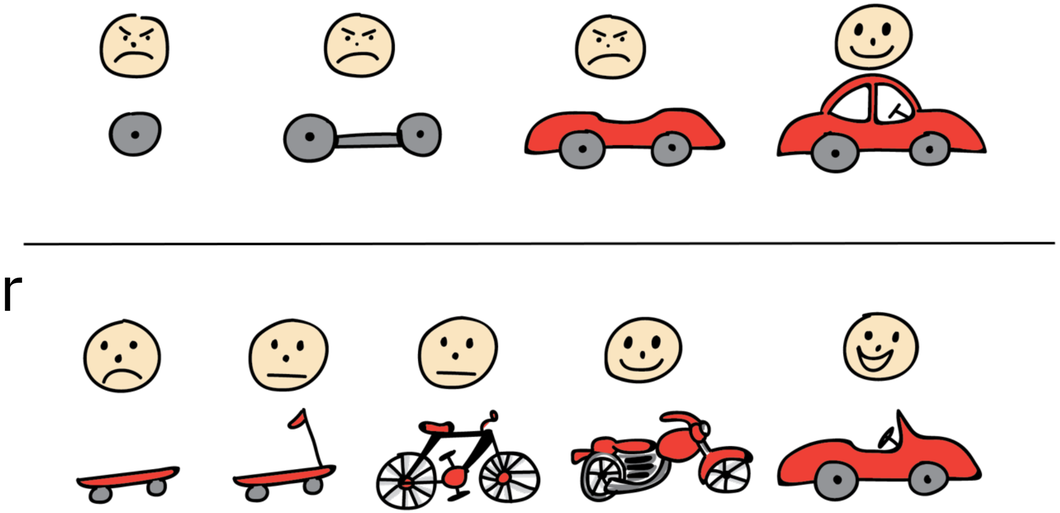
Think Social Coding

- Repositories are **public** and everyone is encouraged to Fork the code and contribute to the project
- **Discuss** the new feature with the repo owner and agree to develop it
- Open an **Issue** and assign it to yourself so that everyone knows what you are working on
- **Fork** the code, create a **branch**, and make your changes
- Issue a **Pull Request** when you are ready to review and merge your work back into the main project



Think Minimum Viable Product

- MVP is NOT the result of "Phase 1" of a project
- It IS the cheapest/easiest thing you can build to start testing your **value hypothesis** and **learning**
- The former focuses on *delivery*, while the latter focuses on *learning*
- At the end of each MVP you decide whether to pivot or persevere

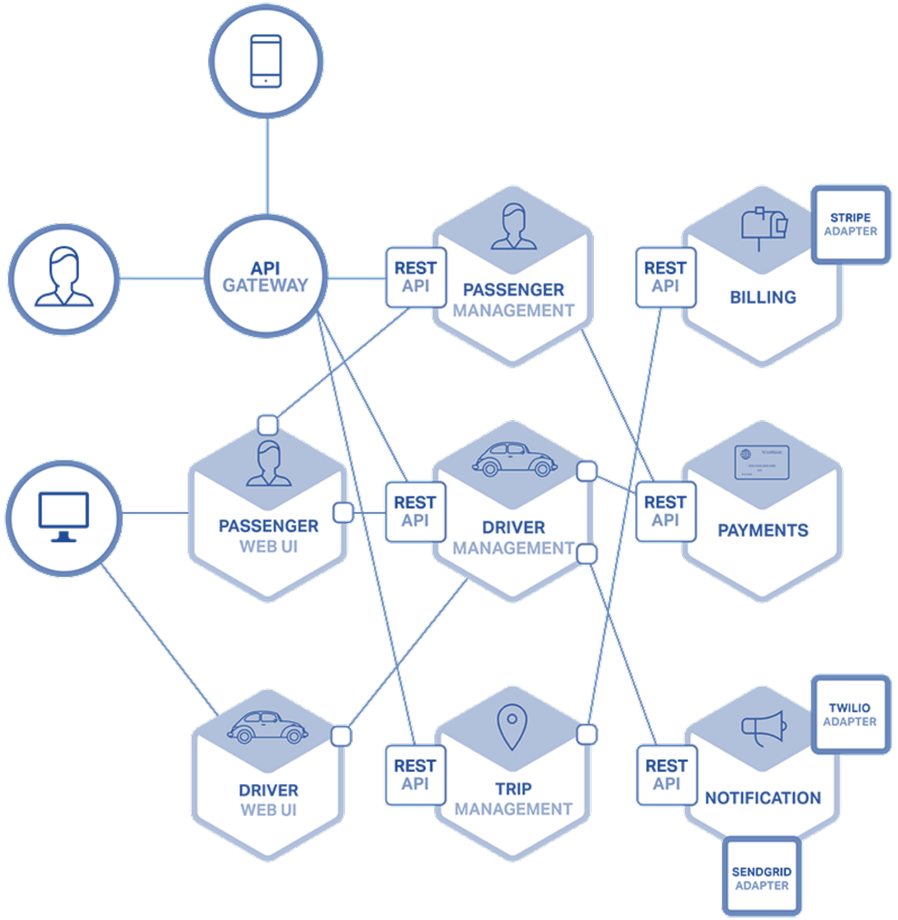


https://blog.crisp.se/2016/01/25/henrikkniberg/making-sense-of-mvp

“To fully leverage DevOps,
you need to think differently about application design.”

Think Cloud Native

- **The Twelve-Factor App** describes patterns for cloud-native architectures which leverage microservices
- Applications are design as a collection of stateless microservices
- State is maintained in separate databases and persistent object stores
- Resilience and horizontal scaling is achieved through deploying multiple instances
- Failing instances are killed and re-spawned, not debugged and patched (cattle not pets)
- DevOps pipelines help manage continuous delivery of services



https://www.nginx.com/blog/introduction-to-microservices/

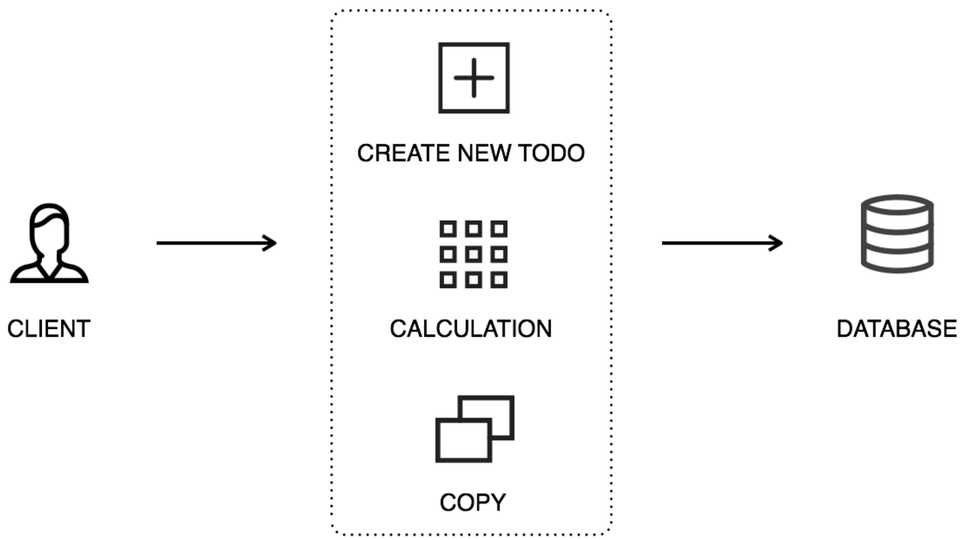
Think Microservices

"...the microservice architectural style is an approach to developing a single application as a **suite of small services**, each **running in its own process** and communicating with lightweight mechanisms, often an HTTP resource API. These services are **built around business capabilities** and **independently deployable** by fully automated deployment machinery."

- Martin Fowler and James Lewis

Monolith vs Microservices Architecture

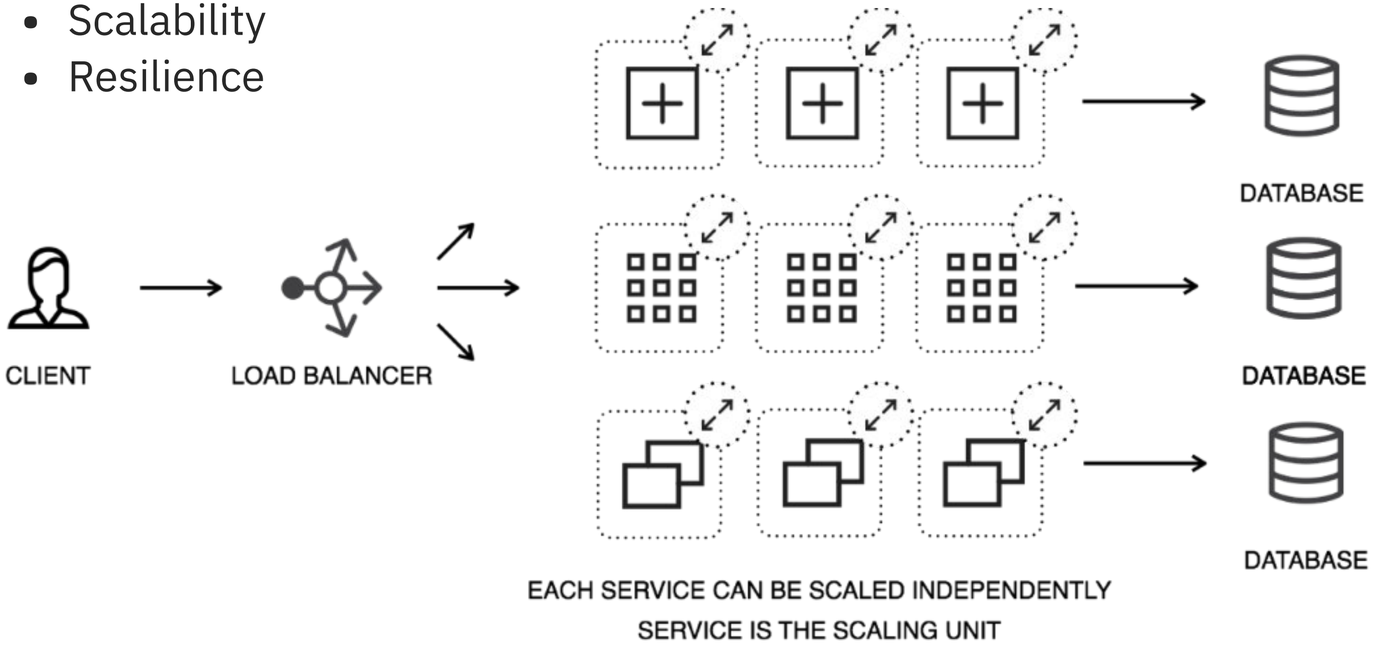
Monolith



All services combined into one build,
written in the same language and application framework

Microservices

- Agility
- Scalability
- Resilience

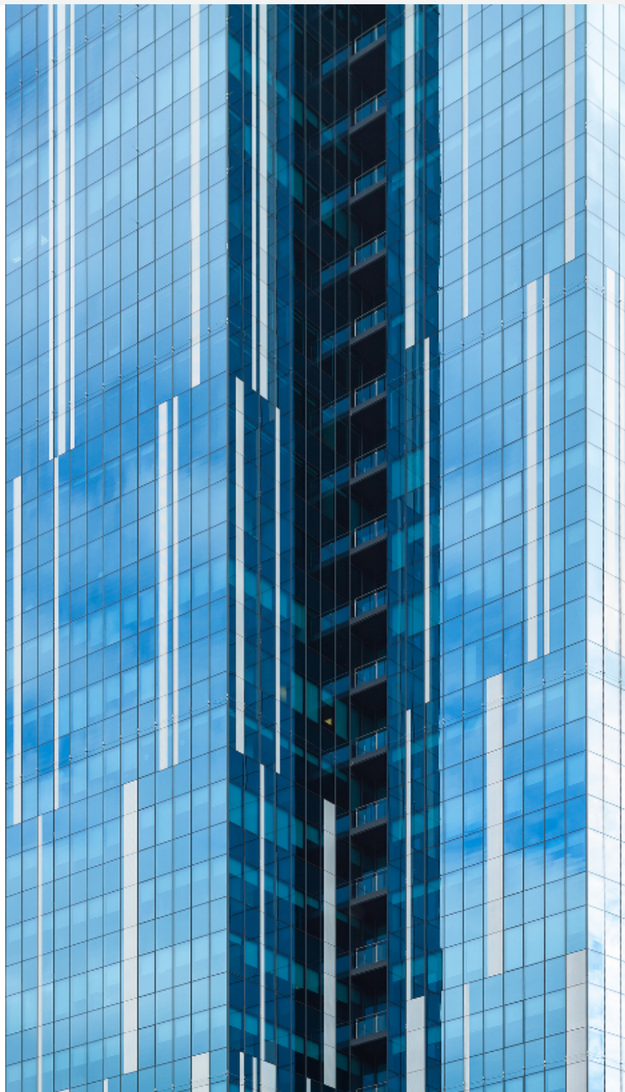


A microservice is a granular, decoupled component within a broader application

“Since failure is inevitable, architect your software to be resilient and to scale horizontally.”

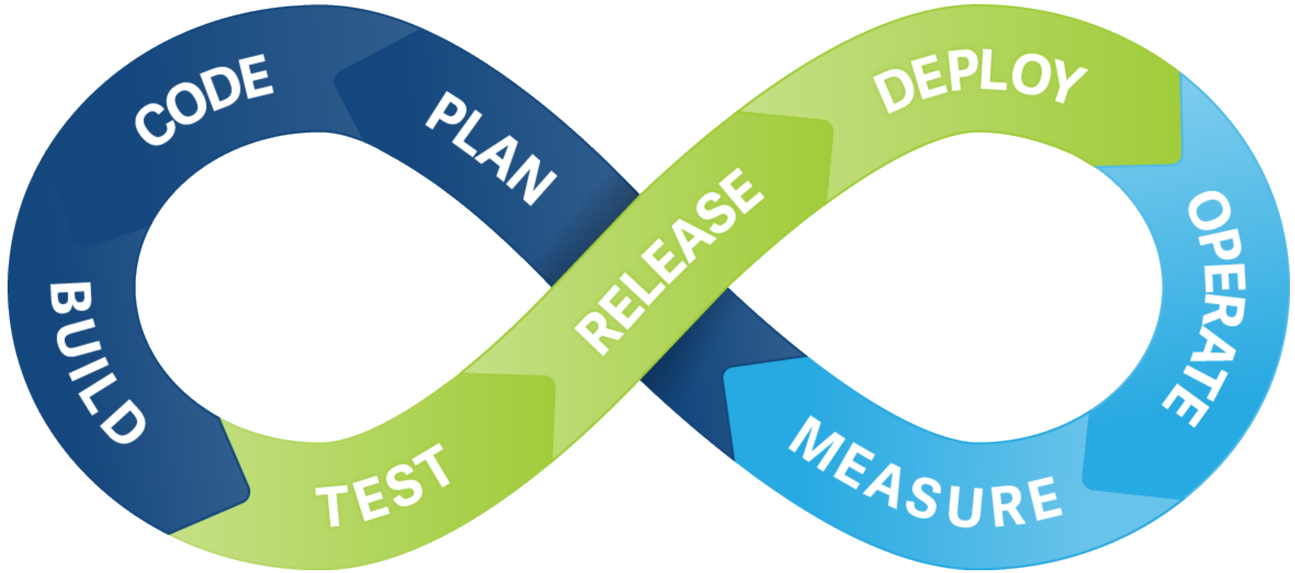
Design for Failure

- **Embrace failures** - they will happen!
  - Move from: How to avoid —> How to identify & what to do about it
  - Move from: Pure operational concern —> developer concern
- External calls to other services that you don't control are especially prone to problems
  - Use separate thread pools
  - Time out quickly
- **Circuit breaker pattern** – identify problem and do something about it to avoid cascading failures
- **Bulkhead pattern** - Isolation from start to limit scope of failure (separate thread pools)
- **Monkey testing** – test by breaking (yes, on purpose! see: Netflix Chaos Monkey and Simian Army)



Think Continuous Automation

- Continuous Integration (CI)
- Continuous Delivery (CD)
- Build Automation
- Canary Rollouts / Blue-Green
- Failing Forward
- Application Release Automation



Automation is not just about speed... it's about repeatability.

Case Study: Knight Capital

- The Knight Capital Group was an American global financial services firm engaging in market making, electronic execution, and institutional sales and trading
- With its high-frequency trading algorithms Knight was:
  - The largest trader in U.S. equities
  - with a market share of 17.3% on NYSE
  - and 16.9% on NASDAQ

Case Study: Knight Capital
(continued)

- Upon deployment, the new **Retail Liquidity Program** (RLP) code in SMARS was intended to replace unused code in the relevant portion of the order router.
- This **unused code** previously had been used for functionality called “Power Peg,” which Knight had discontinued using many years earlier.
- Despite the lack of use, the Power Peg functionality **remained present and callable** at the time of the RLP deployment.
- The new RLP code also **repurposed a flag** that was formerly used to activate the Power Peg code.
- Knight **intended to delete** the Power Peg code so that when this flag was set to “yes,” the new RLP functionality—rather than Power Peg—would be engaged.

Knight Capital's $460 Million Release Failure

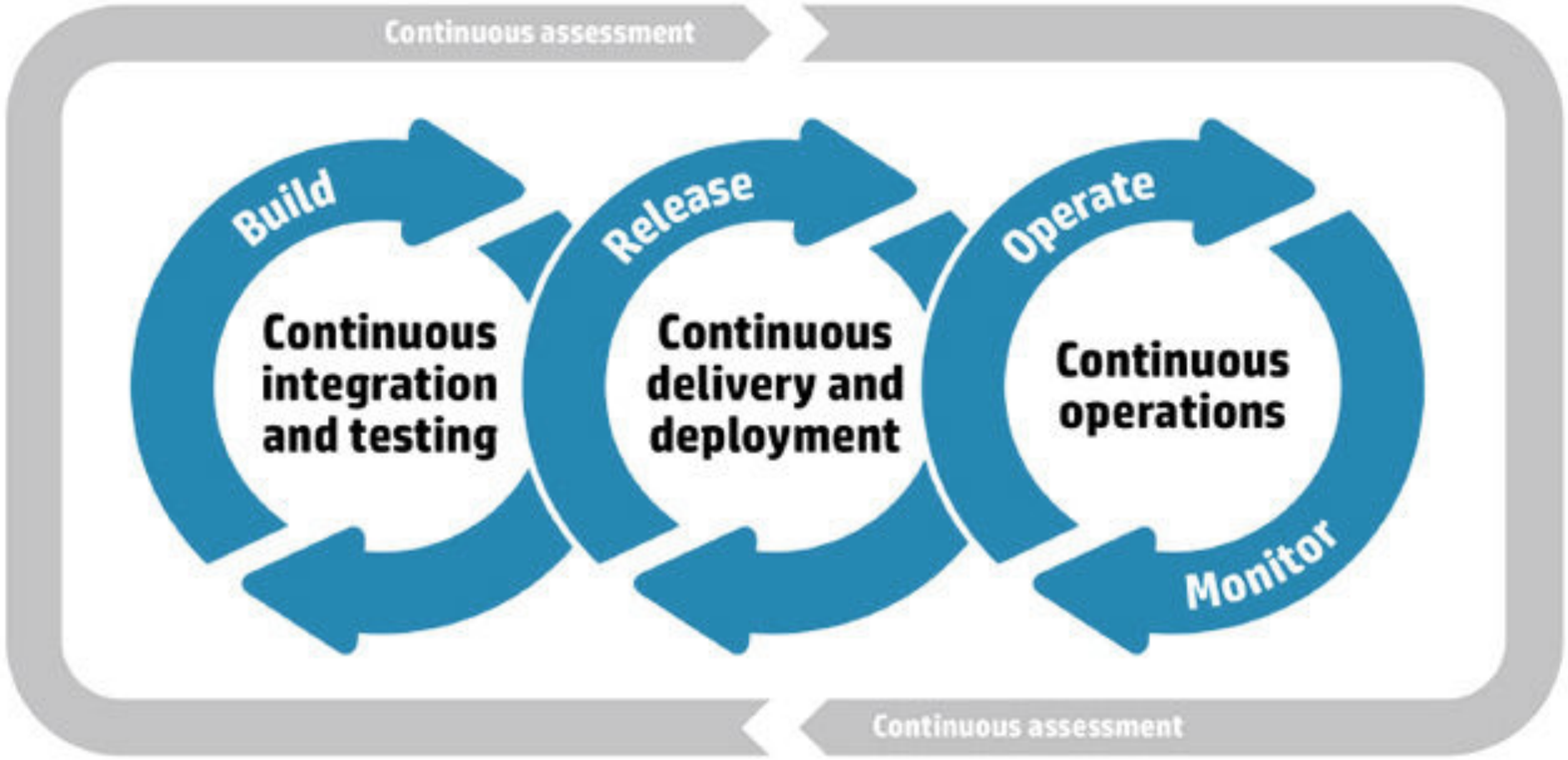
15. Beginning on July 27, 2012, Knight deployed the new RLP code in SMARS in stages by placing it on a limited number of servers in SMARS on successive days. During the deployment of the new code, however, one of Knight's technicians did not copy the new code to one of the eight SMARS computer servers. Knight did not have a second technician review this deployment and no one at Knight realized that the Power Peg code had not been removed from the eighth server, nor the new RLP code added. Knight had no written procedures that required such a review.

Events of August 1, 2012

16. On August 1, Knight received orders from broker-dealers whose customers were eligible to participate in the RLP. The seven servers that received the new code processed these orders correctly. However, orders sent with the repurposed flag to the eighth server triggered the defective Power Peg code still present on that server. As a result, this server began sending child orders to certain trading centers for execution. Because the cumulative quantity function had been moved, this server continuously sent child orders, in rapid sequence, for each incoming parent order without regard to the number of share executions Knight had already received from trading centers. Although one part of Knight's order handling system recognized that the parent orders had been filled, this information was not communicated to SMARS.

17. The consequences of the failures were substantial. For the 212 incoming parent orders that were processed by the defective Power Peg code, SMARS sent millions of child orders, resulting in 4 million executions in 154 stocks for more than 397 million shares in approximately 45 minutes. Knight inadvertently assumed an approximately $3.5 billion net long position in 80 stocks and an approximately $3.15 billion net short position in 74 stocks. Ultimately, Knight realized a $460 million loss on these positions.

Think Continuous Improvement



Actually it's a continuous pipeline that exists in a giant feedback loop

“Being able to recover quickly from failure is more important than having failures less often.”

–John Allspaw, CTO at Etsy

How do you change a culture?

You must change the way people work

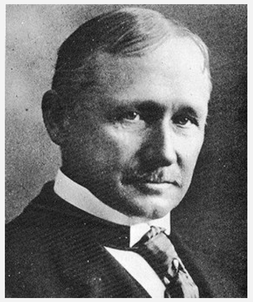
Working DevOps

- Facilitate a culture of **teaming and collaboration**
- Establish **agile development** as a shared discipline
- **Automate relentlessly** to enable rapid DevOps response
- Push **smaller releases faster**, measure and remediate impact

DevOps

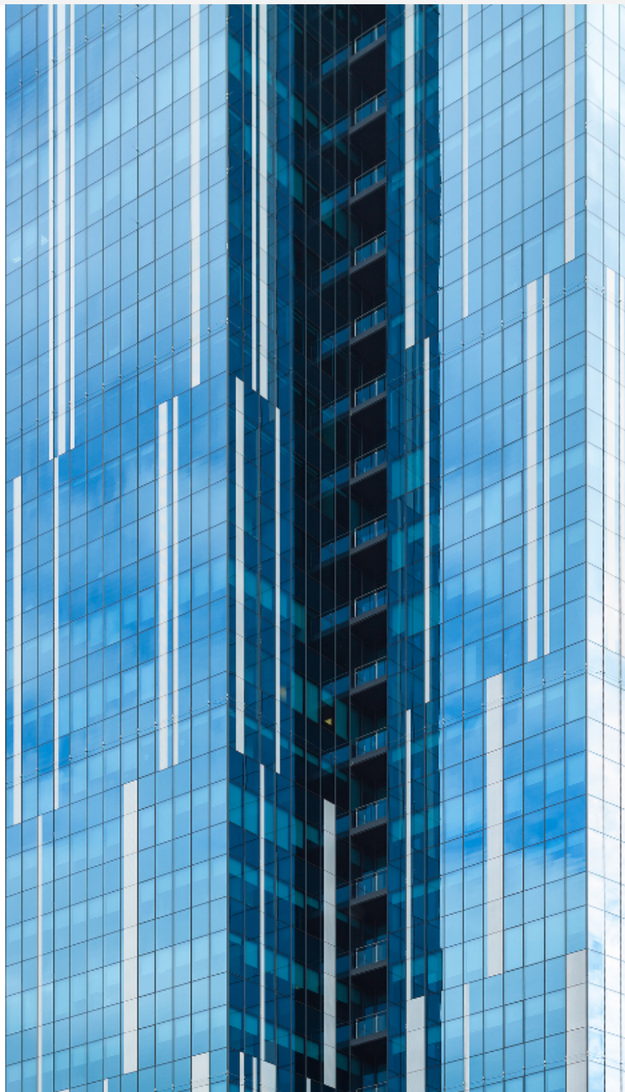
We have worked the same way since the industrial revolution

Taylorism



Named after the US industrial engineer **Frederick Winslow Taylor** (1856-1915) who in his 1911 book ***'Principles Of Scientific Management'*** laid down the fundamental principles of large-scale manufacturing through **assembly-line** factories.

- Adoption of **Command and Control Management**
  - The dominant method of management in the Western world
- Organizations divided into (ostensibly) independent **functional silos**
  - Workers are separated into task specific roles for greater efficiency
- **Decision-making** is separated from work
  - Managers do the planning and decide what workers should do
  - Workers mindlessly do the tasks they are ask to accomplish

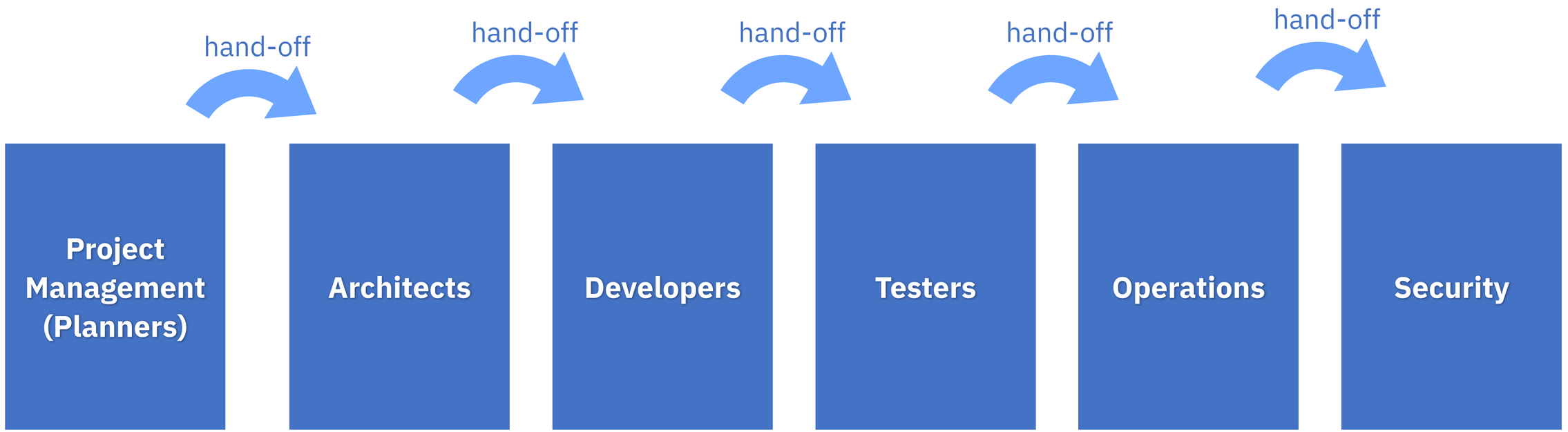


General George Patton

“Never tell people how to do things. Tell them what to do and let them surprise you with their ingenuity.”

Impact of Taylorism on Information Technology

Optimized roles may work great making cars (assembly line), not so good for software development (yields waterfall / silos)



Software Development is NOT like Assembling Automobiles

In software development, most of the parts don't even exist yet

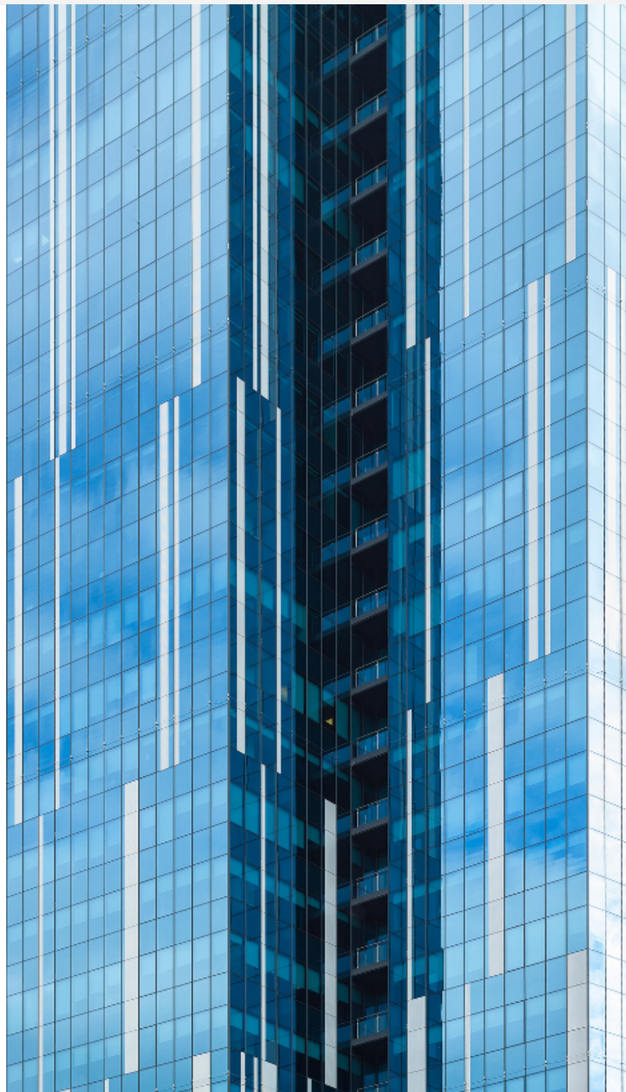
1928 Ford Rouge Complex Model A assembly line negative from the Ford Motor Company archives.

DevOps represents the reincarnation of "craft work"

...because software development is always bespoke

Taylorism is not appropriate for Craft Work

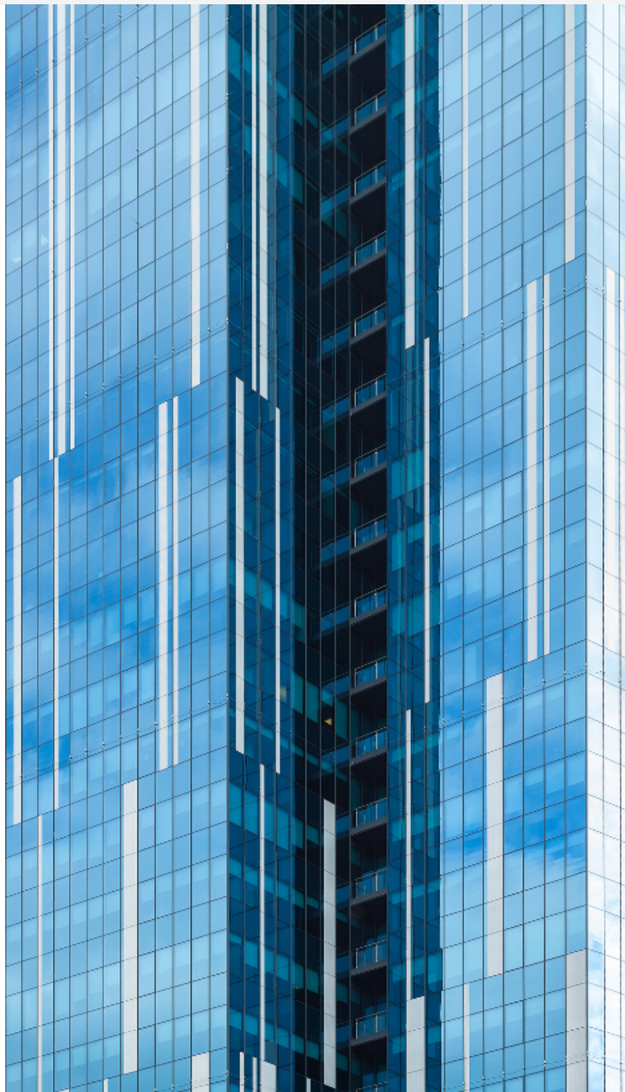
- Taylorism may have been good during the industrial revolution, but not so much in the technology revolution
- Automation has resolved many problems which Taylor sought to address
- The people power requirements have been lowered
- Taylorism is not appropriate for "knowledge" work like software development



Software Engineering vs Civil Engineering

The project model is fundamentally flawed

- **Software development efforts are usually run as if they were civil engineering projects**
  - When a project is complete, the system gets tossed over the wall to operations to run and maintain as part of a "business as usual" effort
  - All the people in the project team get reallocated to new work
- **The project model is fundamentally flawed as a way of doing software development**
  - Software development should be treated as product development instead



https://continuousdelivery.com/2012/10/theres-no-such-thing-as-a-devops-team/

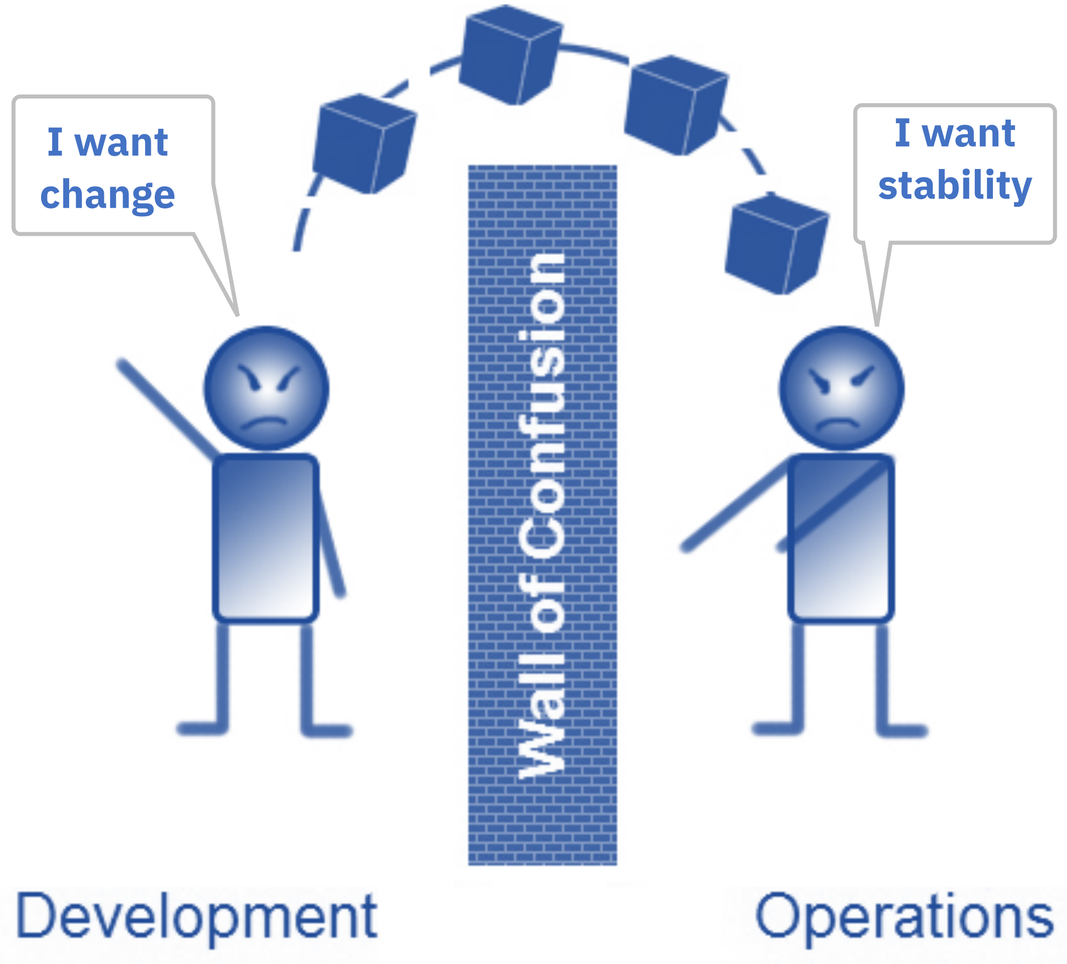
A Clash of Work Cultures

Traditional Ops		DevOps
Manual configuration changes to critical infrastructure	→	Automated deployment to all environments
Application architectures defined by network design	→	Network design defined by application architectures
Bespoke infrastructure built once, then maintained	→	Ephemeral infrastructure created for each new deployment
Risk managed through change windows	→	Risk managed through progressive activation
Process biased toward “build once”	→	Processes re-engineered for high volume, rapid throughput of changes

The Kobayashi Maru

...the no-win scenario

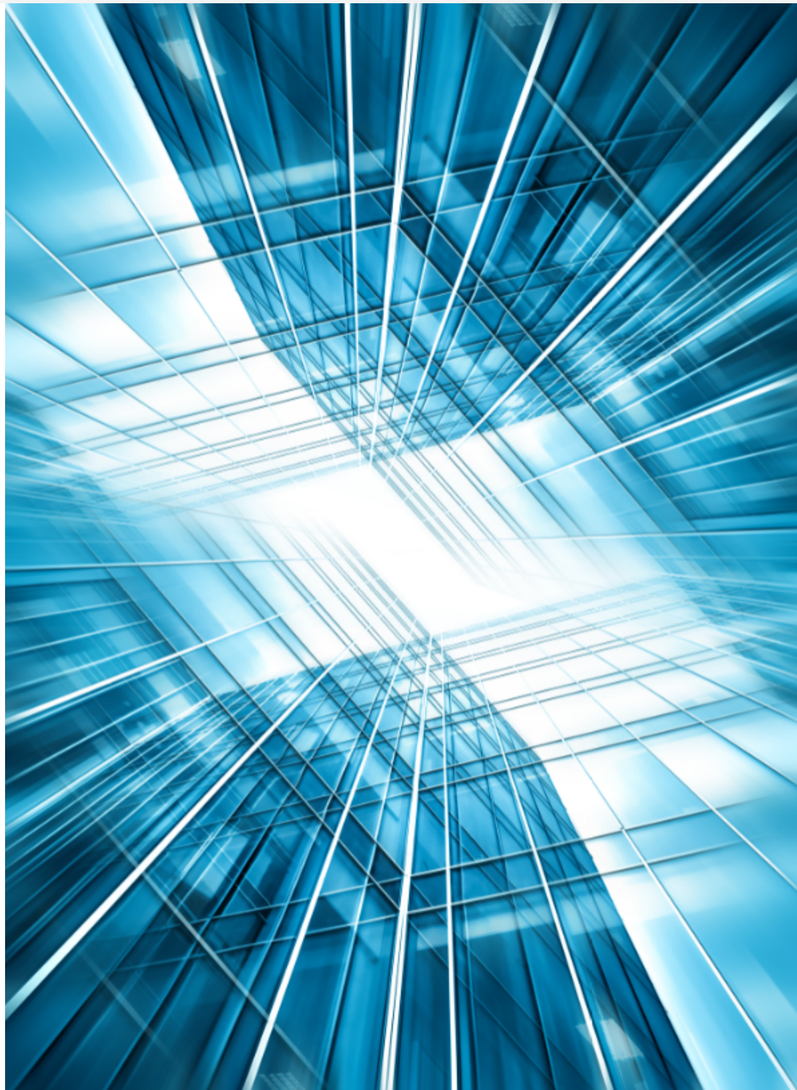
- **Development wants Innovation**
  - Keep up users' changing needs by developing and deploying new and enhanced capabilities.
- **Operations want Stability**
  - Ensure that users can use those services and applications and that their data and information is kept safe.



Source: Andrew Clay Shafer, Senior Director of Technology at Pivotal

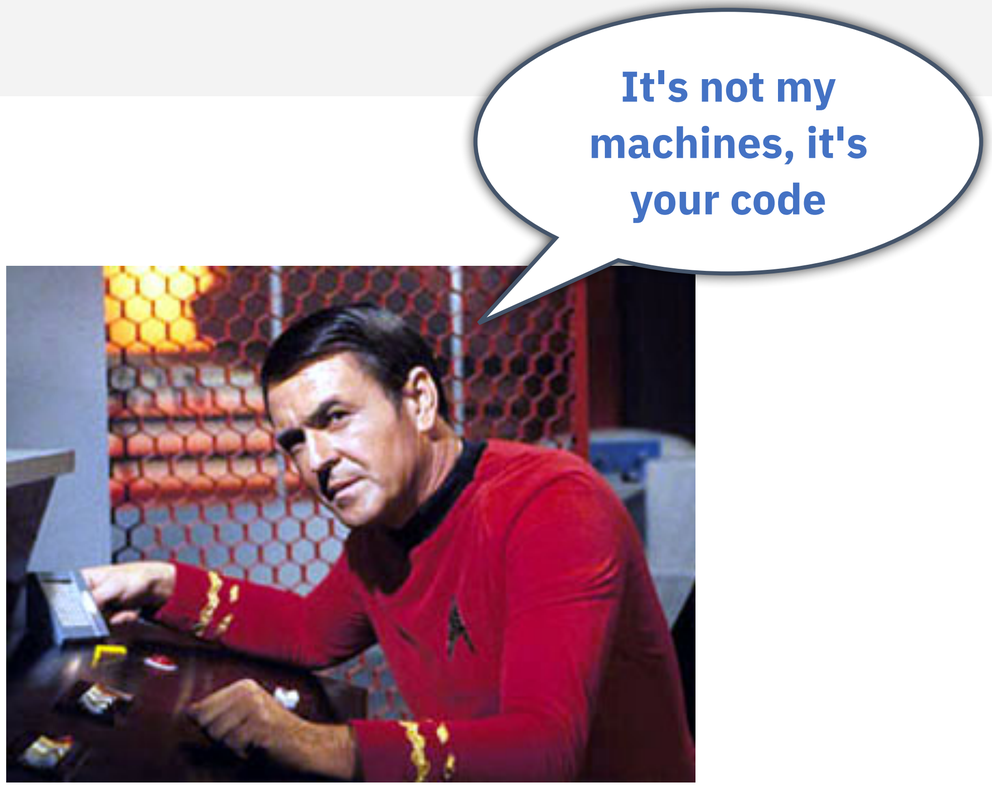
Diametrically Opposed Views

- **Enterprises** are built around end-to-end processes that see “new” as complex, high risk, expensive and time consuming
  - To be delivered as a one-time project
- **DevOps** seeks to turn this on its head
  - DevOps delivers a continual series of small changes
  - These cannot survive traditional overheads



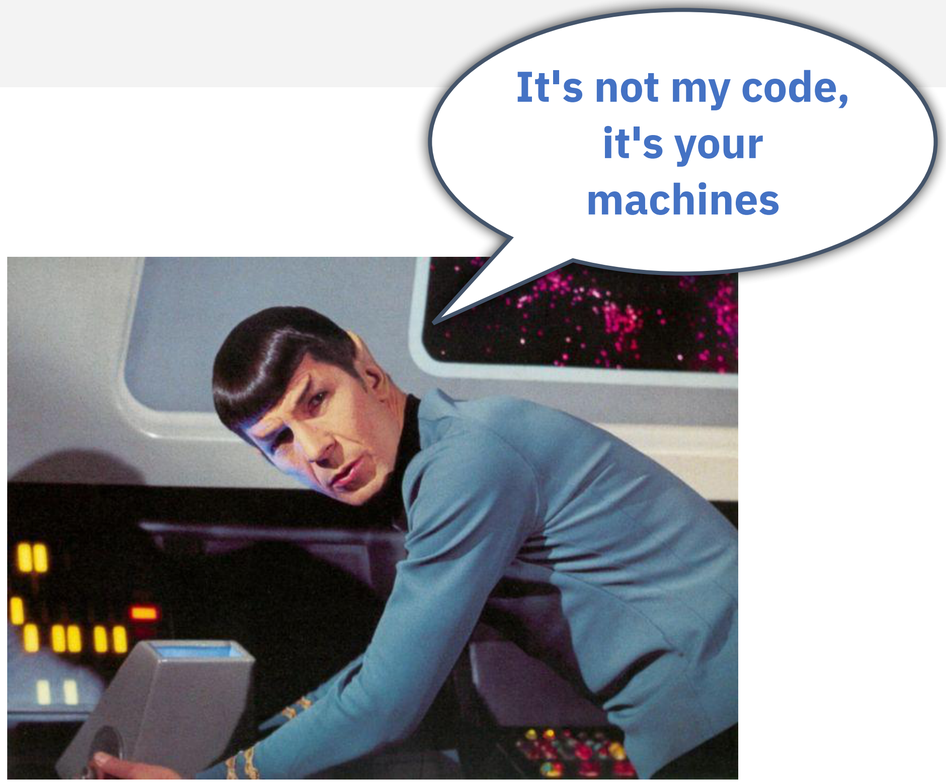
Operations View of Development

- Development teams throw dead cats over the wall
- Manually implemented changes
- Lack of back-out plans
- Lack of testing
- Environments that don’t look like Production



Development View of Operations

- All-or-nothing changes
- Change windows in the dead of night
- Implemented by people furthest away from the application
- Using cut-and-paste from Word documents called “run-books”



IT Silos Breed Apathy

“If the web site works, the developers get the praise

If the web site is down, operations gets the blame!”

...did I mention the Kobayashi Maru?

Required DevOps behaviors

DevOps turns a number of traditional IT practices on their head

Organizational silos and hand-offs > Shared ownership and high collaboration

Fear of change > Risk management by embracing change

Build once, hand crafted "snow flakes" > Ephemeral infrastructure as code

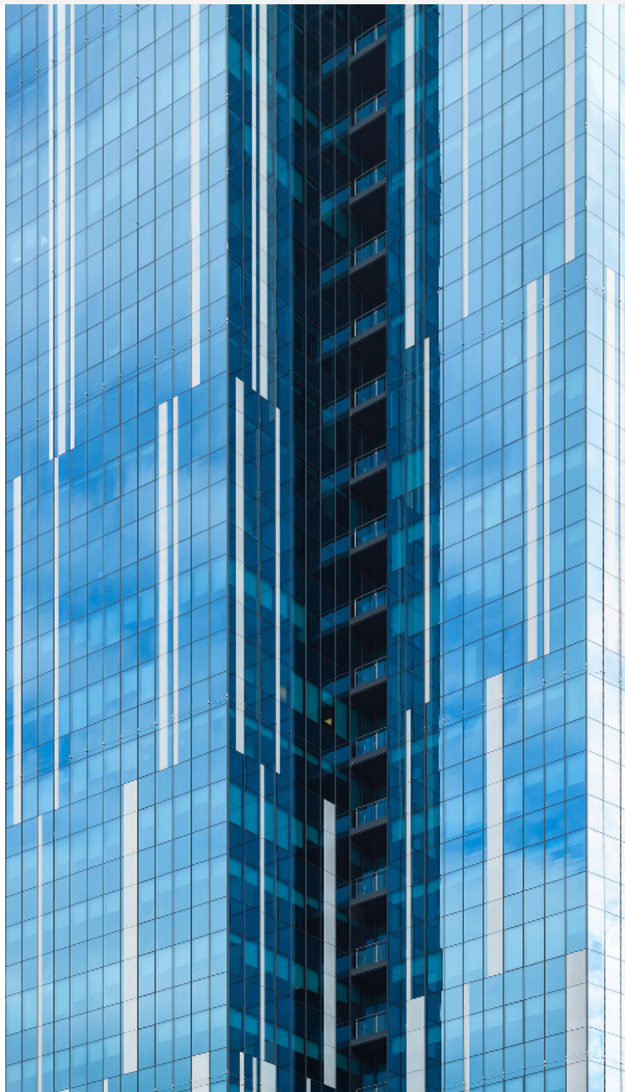
Manual fulfillment > Automated Self Service

Alarms, call-backs, and escalations > Feedback loops and data driven responses

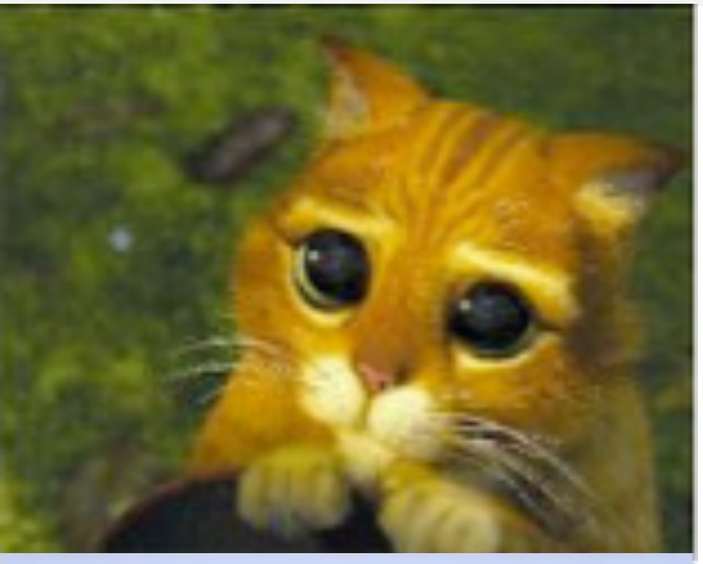
How DevOps Manages Risk

DevOps manages risk through increasing the rate of change rather than aversion to it

- **Deployment is king**
  - Deployment must be painless
  - You have deployed the same thing several times before it gets to production
- **Deployment is decoupled from activation**
  - Risk is managed via activation controls (blue-green deploys with canary testing)
    - 10% of the user base, waves of activation, etc.
- **Deployment is not “one size fits all”**
  - It is a rail yard of interconnecting steps and microprocesses



Servers are Cattle not Pets



- Pets are given names like pussinboots.cern.sh
- They are unique, lovingly hand raised and cared for
- When they get ill, you nurse them back to health

- Cattle are given numbers like vm0042.cern.ch
- They are almost identical to other cattle
- When they get ill, you get another one

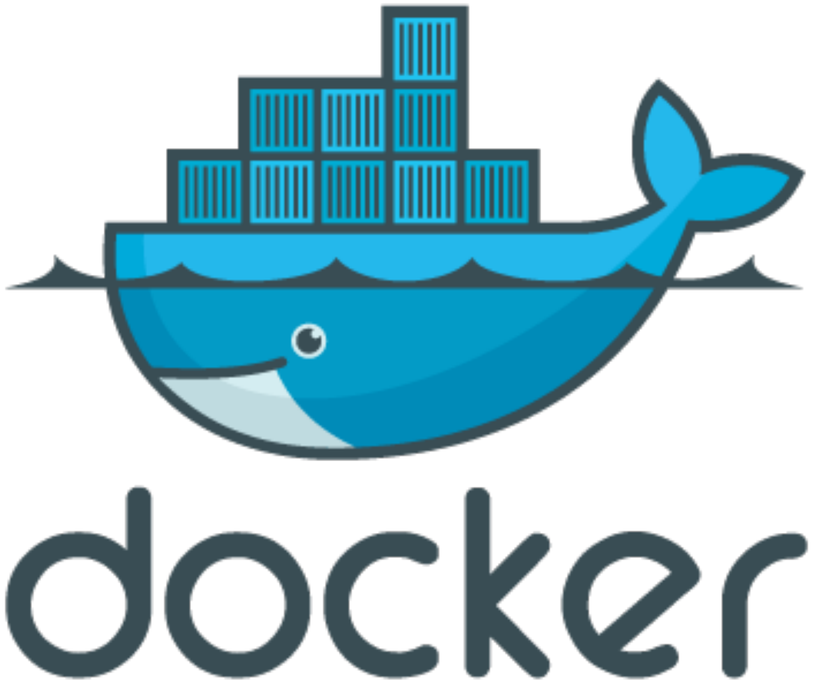
Ephemeral Infrastructure

The Cloud has enabled new ways of managing server builds, software releases, and server entropy

- **Cattle, not Pets**
  - Servers are built on demand via automation
  - Logging into the server is seen as failure
- **Release through parallel infrastructure**
  - Build the new version on new infrastructure; stage transition between environments (zero downtime)
- **Transient Infrastructure**
  - Throw away when it is no longer needed
  - This eliminates entropy - a major source of failure

Immutable Delivery

- Applications are packaged in containers
- Same container that developer runs on their laptop runs in production
- Rolling updates with immediate roll-back
- No variance limits side-effects
- Dependencies are contained



Zero-Downtime Deployments

- Blue-green deployment is a zero-downtime deployment technique that consists of two nearly identical production environments, called Blue and Green.
- They differ by the artifacts that the developer has intentionally changed, typically by the version of the application. At any given time, at least one of the environments is active.
- Using the blue-green deployment technique, you can realize the following benefits:
  - Take software quickly from the final stage of testing to live production.
  - Deploy a new version of an application without disrupting traffic to the application.
  - Rollback rapidly. If there is something wrong with one of your environments, you can quickly switch to the other environment.

“DevOps is about breaking down the silos and working as a Single Agile Team.”

Everyone Working Agile

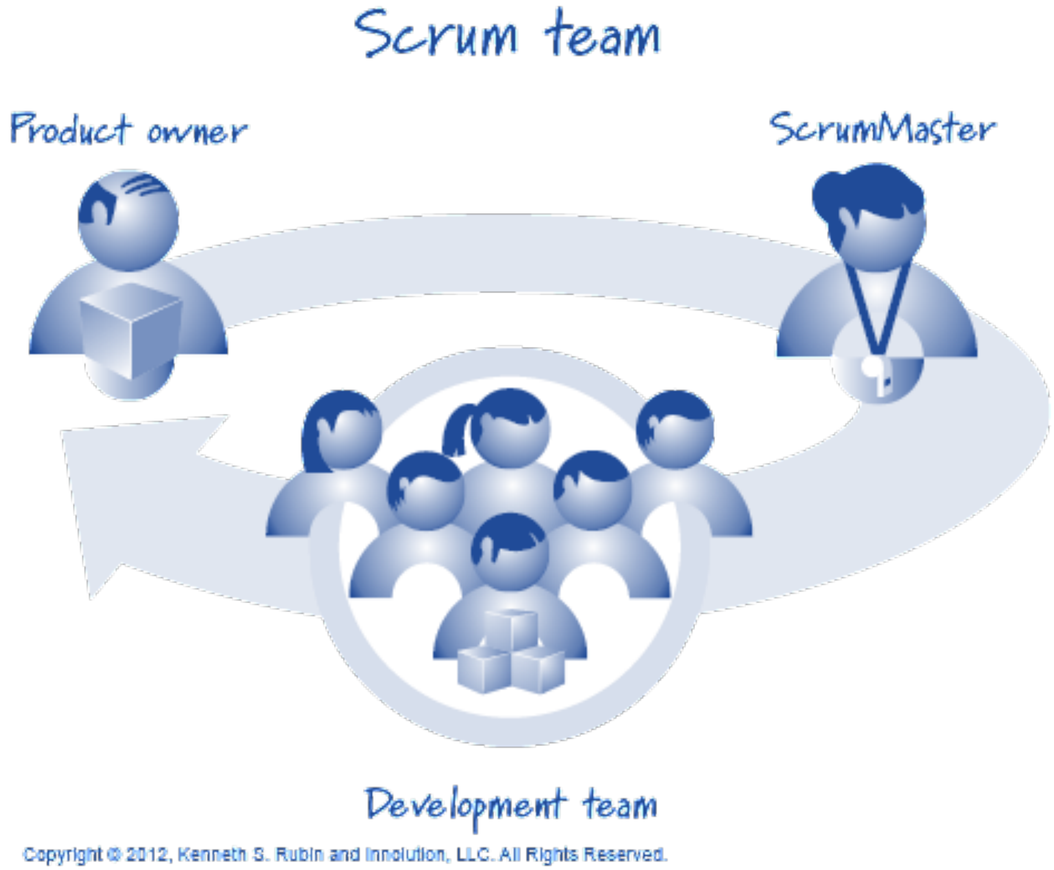
Agile Manifesto

We are uncovering better ways of developing
software by doing it and helping others do it.
Through this work we have come to value:

Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan

That is, while there is value in the items on the right,
we value the items on the left more.

How Agile are your teams?



- ☑ Small team (5 +/- 2)
- ☑ Dedicated
- ☑ Co-located
- ☑ Cross-functional
- ☑ Self managing

Agile Antipatterns

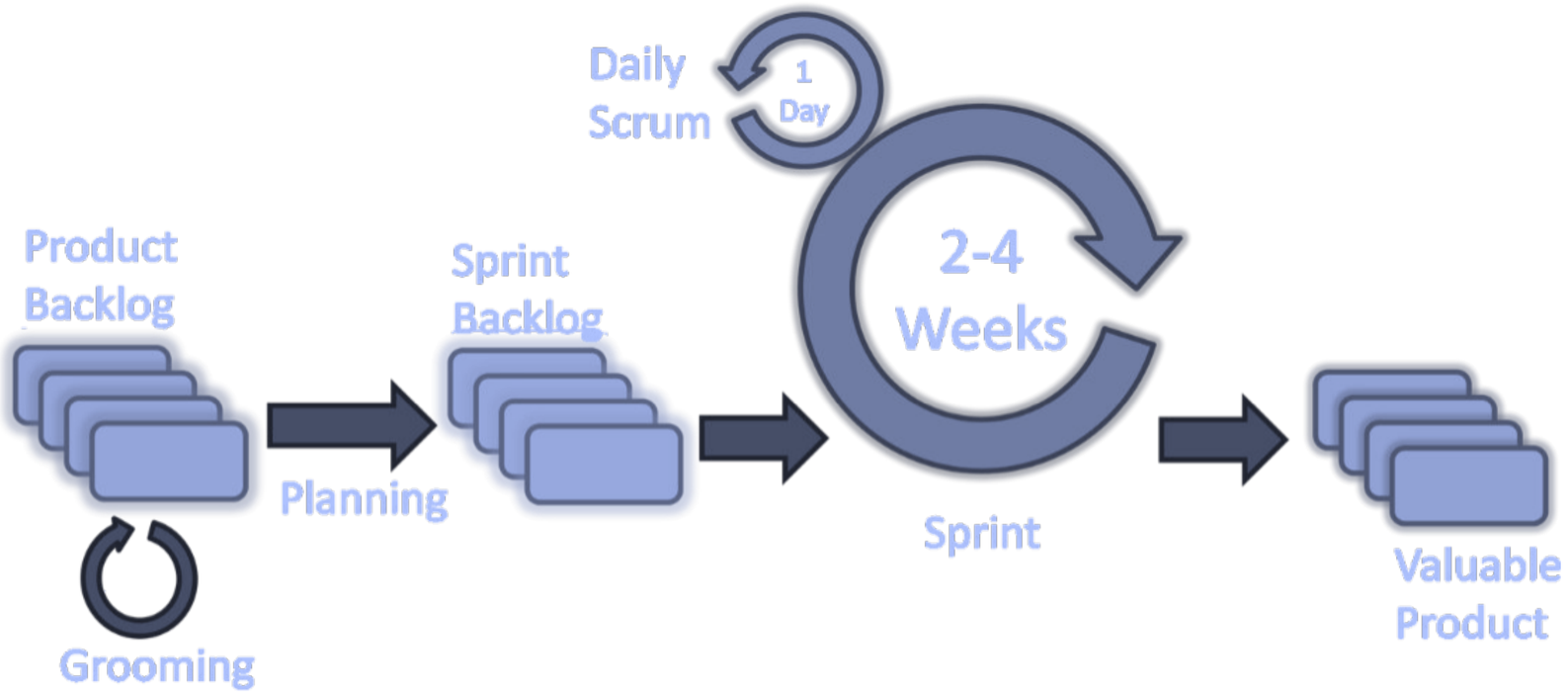
- Lack of real Product Owner
- If your teams are too large
- If your teams and not dedicated
- If your teams are geographically distributed
- If you teams siloed
- If your teams are not self managing

YOU WILL FAIL!!!

...and you should not wonder why

Working as an Agile Team

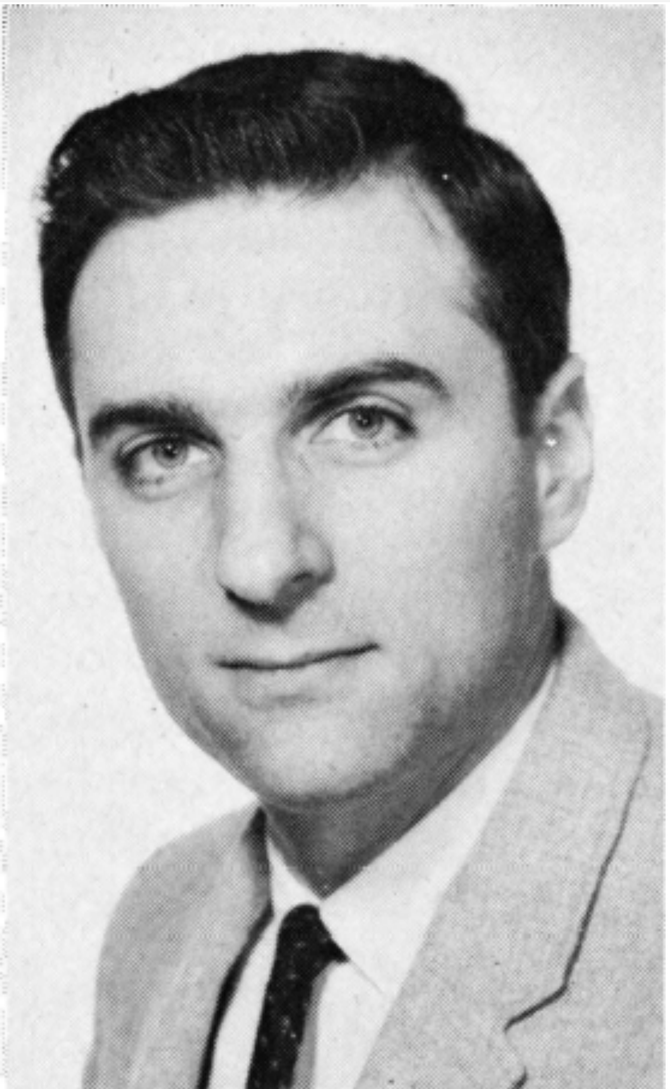
- Iterative Sprints
- Groomed Backlogs
- Customer Stories
- 2 Week Deliverables



How do you change a culture?

You must change the way people are organized

Conway’s Law



Any organization that designs a system (defined broadly) will produce a design whose structure is a copy of the organization's communication structure

- Melvin Conway, Datamation, 1968

e.g., if you ask an organization with 4 teams to write a compiler... you will get a 4-pass compiler!

http://www.melconway.com/Home/Conways_Law.html

Traditional Organization around Technology (Bad)

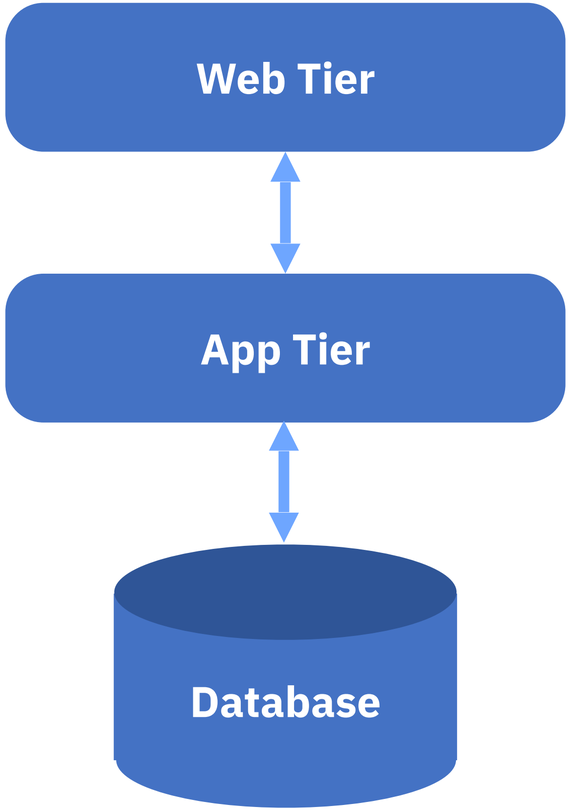
Organization Structure

User Interface Team

Application Team

Database (DBA) Team

Application Structure



Organization around Business Domains (Good)

Login
Registration
Users

Account Team

Personalization

Personalization Team

Inventory
Shipping
Receiving

Warehouse Team

“If you are not going to organize around business domains, you are not going to get the full benefit of DevOps”

Spotify Engineering Culture

https://www.youtube.com/watch?v=R2o-Xm3UVjs

Spotify® Engineering Culture

Rules are a good start, then break them when needed

Agile > Scrum

Principles > Practices

Servant Leaders > Process Master

Community > Structure

Enable > Serve

Trust > Control

Failure Recovery > Failure Avoidance

Impact > Velocity

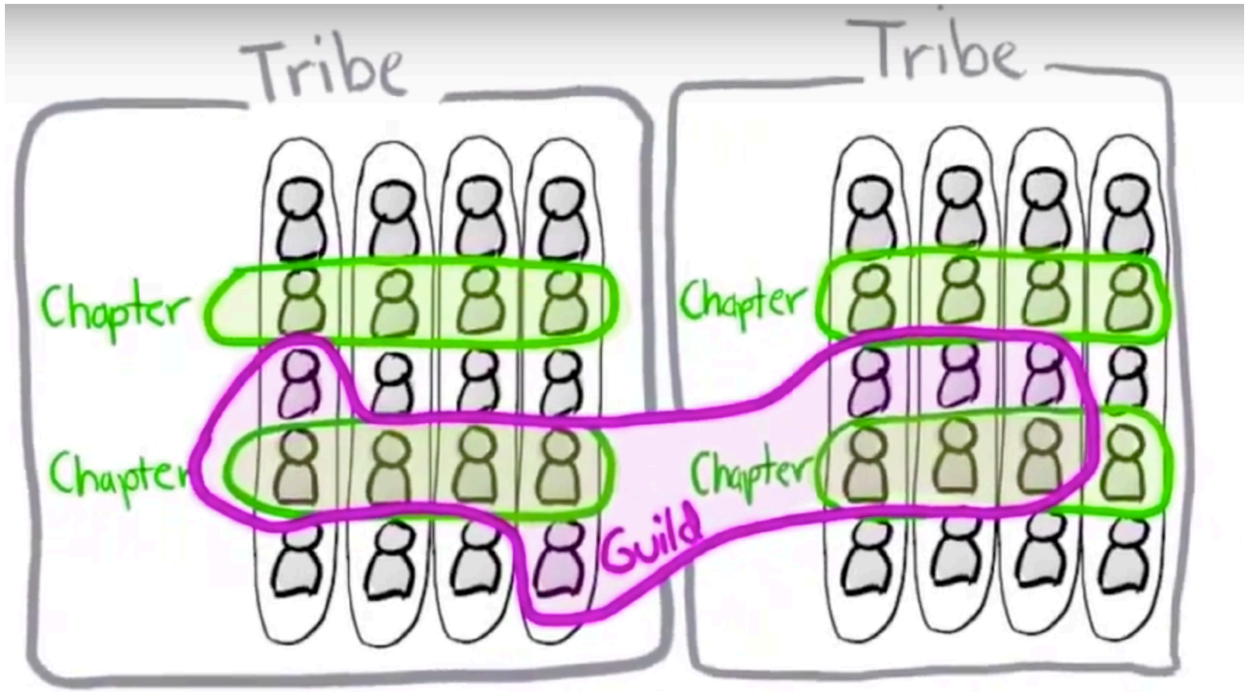
Innovation > Predictability

Value Delivery > Plan Fulfillment

Chaos > Bureaucracy

Spotify® Organizational Structure

- **Squads** are grouped into Tribes (light-weight matrix)
- **Chapters** of competency areas are formed across Squads
- **Guilds** are informal light-weight community of interests across the company



Spotify® Autonomous Squads

Loosely coupled, Tightly aligned squads

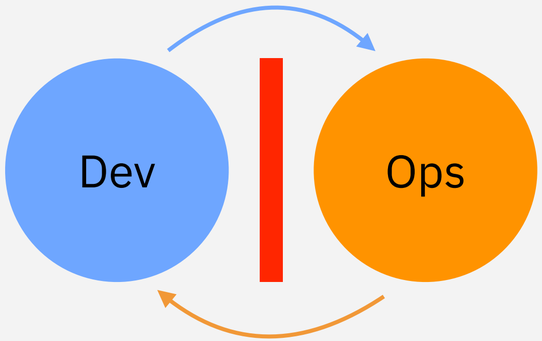
- Each Squad has it’s own mission aligned with the business
  - Feels like a "mini-startup"
  - Self Organizing / Cross-functional
  - 5-7 engineers, less than 10
- Squads have end-to-end responsibility for what they build
  - Build, commit, deploy, maintenance, operations, EVERYTHING!!!
  - With a long term mission usually around a single business domain

There's No Such Thing as a "DevOps Team"

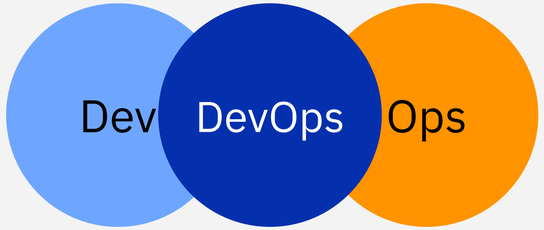
That’s an anti-pattern

Different Perspectives of DevOps

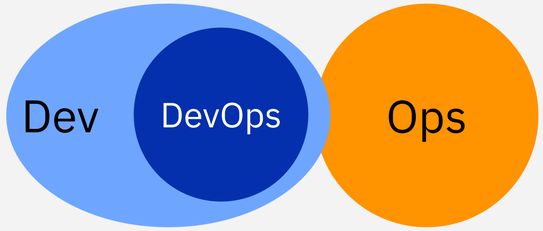
Large Enterprises



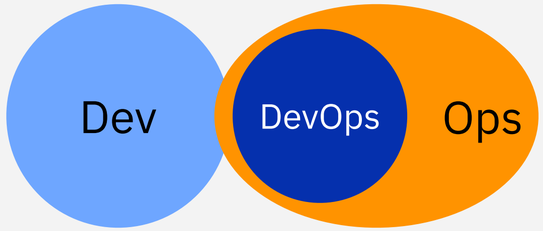
Traditional Development & Operations



DevOps is a separate team



DevOps is something Devs do



DevOps is something Ops does

Cloud Native Startups

DevOps is the company culture

“The DevOps movement addresses the dysfunction that results from organizations composed of functional silos. Thus, creating another functional silo that sits between dev and ops is clearly a poor (and ironic) way to try and solve these problems.”

– Jez Humble, The DevOps Handbook

https://continuousdelivery.com/2012/10/theres-no-such-thing-as-a-devops-team/

DevOps Organizational Objective

Shared Consciousness

...with

Distributed (local) Control

“Bad behavior arises when you abstract people away from the consequences of their actions.”

– Jez Humble

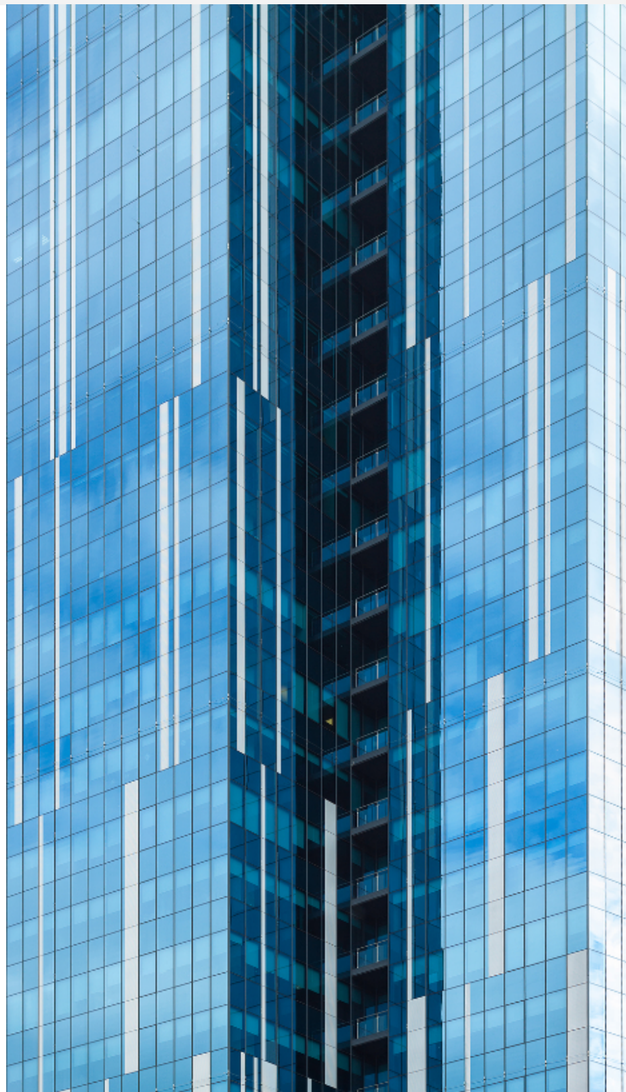
https://continuousdelivery.com/2012/10/theres-no-such-thing-as-a-devops-team/

Functional Silos Breed Bad Behavior

- Bad behavior arises when you abstract people away from the consequences of their actions.
- Functional silos abstract people away from the consequences of their actions.
- For example: By adding a QA Team, developers are abstracted away from the consequences of writing buggy code.

Actions have Consequences

- **Make people aware of the consequences of their actions**
  - Create cross-functional teams - or -
  - Have developers rotate through operations teams
  - Have operations people attend developer standups and showcases
- **Make people responsible for the consequences of their actions**
  - Having developers on Pager Duty, or own the SLA for the products and services they build



https://continuousdelivery.com/2012/10/theres-no-such-thing-as-a-devops-team/

“You build it, you run it!”

–Werner Vogels, CTO of Amazon

How do you change a culture?

You must change the way people are measured

On the folly of rewarding for A, while hoping for B

"Whether dealings with monkeys, rats, or human beings, it is hardly controversial to state that most organisms seek information concerning what activities are rewarded, and then seek to do (or at least pretend to do) those things, often to the exclusion of activities not rewarded. The extent to which this occurs of course will depend on the perceived attractiveness of the rewards offered, but neither operant nor expectancy theorists would quarrel with the essence of this notion."

–Steven Kerr, Ohio State University

Academy of Management Journal, Dec 1975

You get what you measure

Measure what Matters

...because you get what you Measure

- If you measure widget production you will get **lots of widgets**
- If you measure lines of code you will get **many lines of code**
- If you measure people against each other by ranking them, you will get **anti-social behavior**
- But if you measure developers by their social interaction and sharing... you will get **Social Coders!**

DevOps Changes the Objective

- Old School is focused on Mean Time To Failure (MTTF)
  - Make sure you never go down
- DevOps is focused on Mean Time To Recovery (MTTR)
  - You will go down, make sure you can recover quickly!

DevOps Metrics

- A **BASELINE** provides a concrete number for comparison as you implement your DevOps changes:
  - It currently requires six team members 10 hours to deploy a new release of our product.
  - This costs us $X for every release
- Metric **GOALS** allow you to reason about these numbers and judge the success of your transition process:
  - Reduce deployment time from 10 hours to 2 hours.
  - Increase percentage of defects detected in testing from 25% to 50%

Vanity Metrics

...good for feeling awesome, bad for action

- Consider the total number of daily “hits” to your website is 10,000
- Now what? (what does a "hit" represent?)
  - Do you really know what actions you took in the past that drove those visitors to you?
  - Do you really know which actions to take next?
  - In most cases, I don’t think it’s very helpful

Actionable Metrics

- Imagine you add a new feature to your website, and you do it using an A/B split-test in which 50% of customers see the new feature and the other 50% don't.
- A few days later, you take a look at the revenue you've earned from each set of customers, noticing that group B has 20% higher revenue per-customer.
- **Think of all the decisions you can make:**
  - Obviously, roll out the feature to 100% of your customers
  - Continue to experiment with more features like this one and …
  - Realize that you've probably learned something that's particular valuable to your customers.

http://tim.blog/2009/05/19/vanity-metrics-vs-actionable-metrics/

Actionable Metric Examples

- Reduce time-to-market for new features.
- Increase overall availability of the product.
- Reduce the time it takes to deploy a software release.
- Increase the percentage of defects detected in testing before production release.
- Make more efficient use of hardware infrastructure.
- Provide performance and user feedback to the product manager in a more timely manner.

http://tim.blog/2009/05/19/vanity-metrics-vs-actionable-metrics/

Top 4 Actionable Metric

1. **Mean Lead Time**
   - How long does it take from idea to production?
2. **Release Frequency**
   - How often can you deliver changes?
3. **Change Failure Rate**
   - How often to changes fail?
4. **Mean Time to Recovery (MTTR)**
   - How quickly can you recover from failure?

Nicole Forsgren - AWS re:Invent 2017: Tools Won't Fix Your Broken DevOps
https://www.youtube.com/watch?v=gsjCWrCUjNg

Culture Measurements

Measured on a scale of Strongly Agree to Strongly Disagree

- On my team information is actively sought
- On my team failures are learning opportunities and messengers of them are not punished
- On my team responsibilities are shared
- On my team cross functional collaboration is encouraged and rewarded
- On my team failure causes inquiry
- On my team new ideas are welcomed

Nicole Forsgren, CEO & Chief Scientist at DevOps Research and Assessment (DORA)

Cycle Time

- Cycle time is a key metric for Agile kanban teams.
- Cycle time is the amount of time it takes for a unit of work to travel through the team's workflow–from the moment work starts to the moment it ships.
- By optimizing cycle time, the team can confidently forecast the delivery of future work.

Keys to High Performance

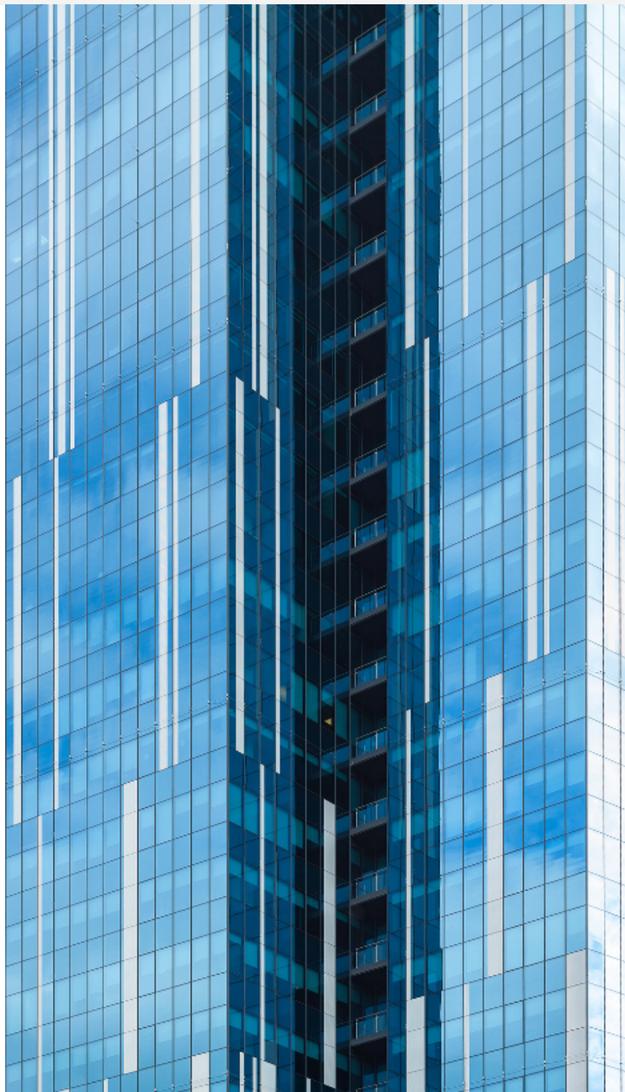
You MUST do all three!

1. Technical Practices
2. Lean Processes (Agile)
3. Culture

“Measurements should encourage innovation and collaboration, and not punish failure”

Busted DevOps Myths

- You cannot buy DevOps In-A-Box
- You cannot order 20 units of DevOps for this quarter
- You cannot sprinkle DevOps on something to make it better
- You cannot become DevOps without changing your culture
  - You can’t change your companies culture just by adopting new tools ...but they can help reinforce it
- Using Containers won't fix your broken culture
- You cannot maintain your current organizational structure and become DevOps



The Entire Organization Must Change

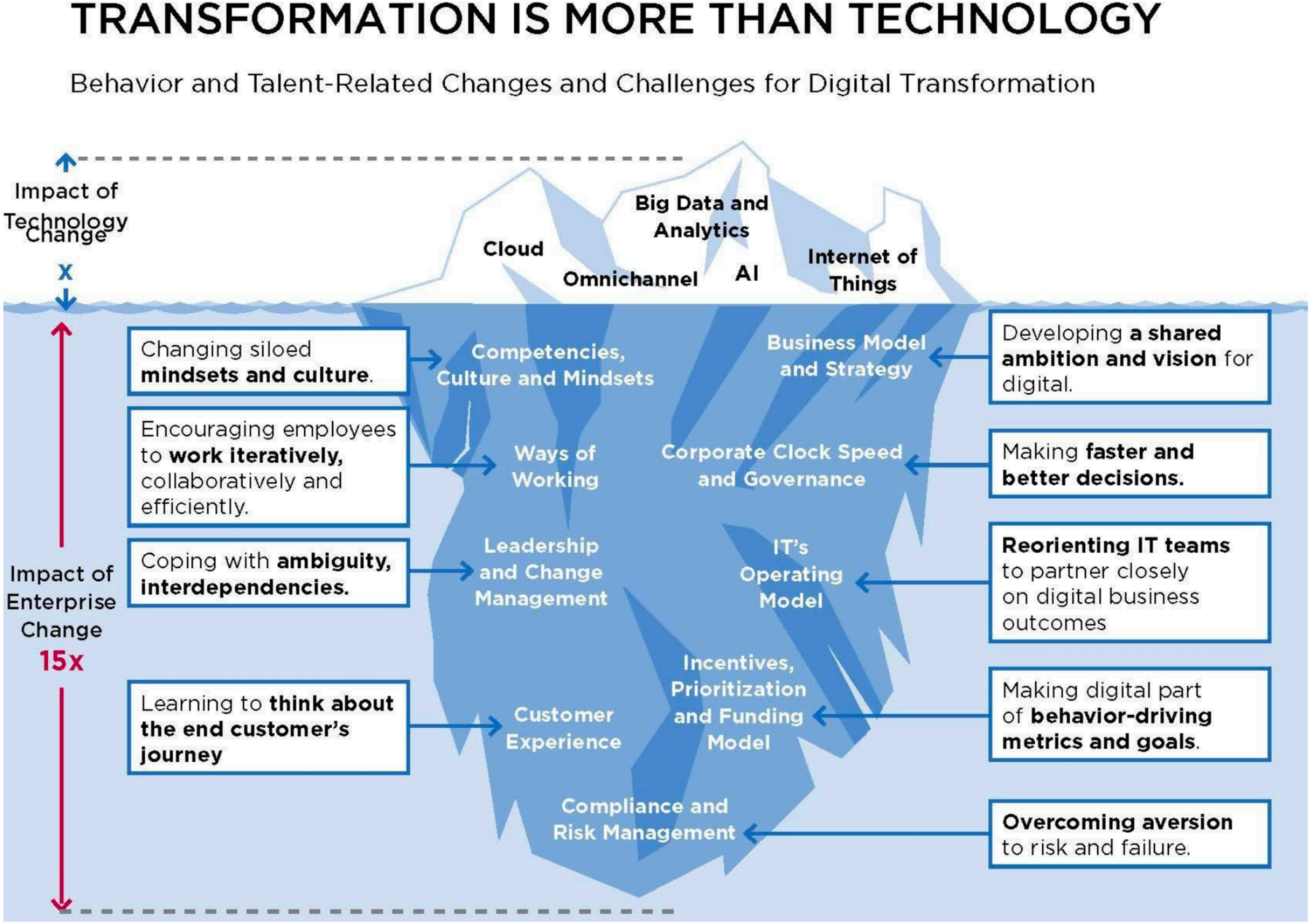
Studies show, that in hierarchical and bureaucratic organizations trying to adopt DevOps on a team level without influencing the environment which supports the change (executive level), people are frequently taught by agile coaches about courage, focus, commitment, respect and openness.

However, the system they work in very often promotes politics, blame games and escalation paths, which are in direct contradiction to the aforementioned values.

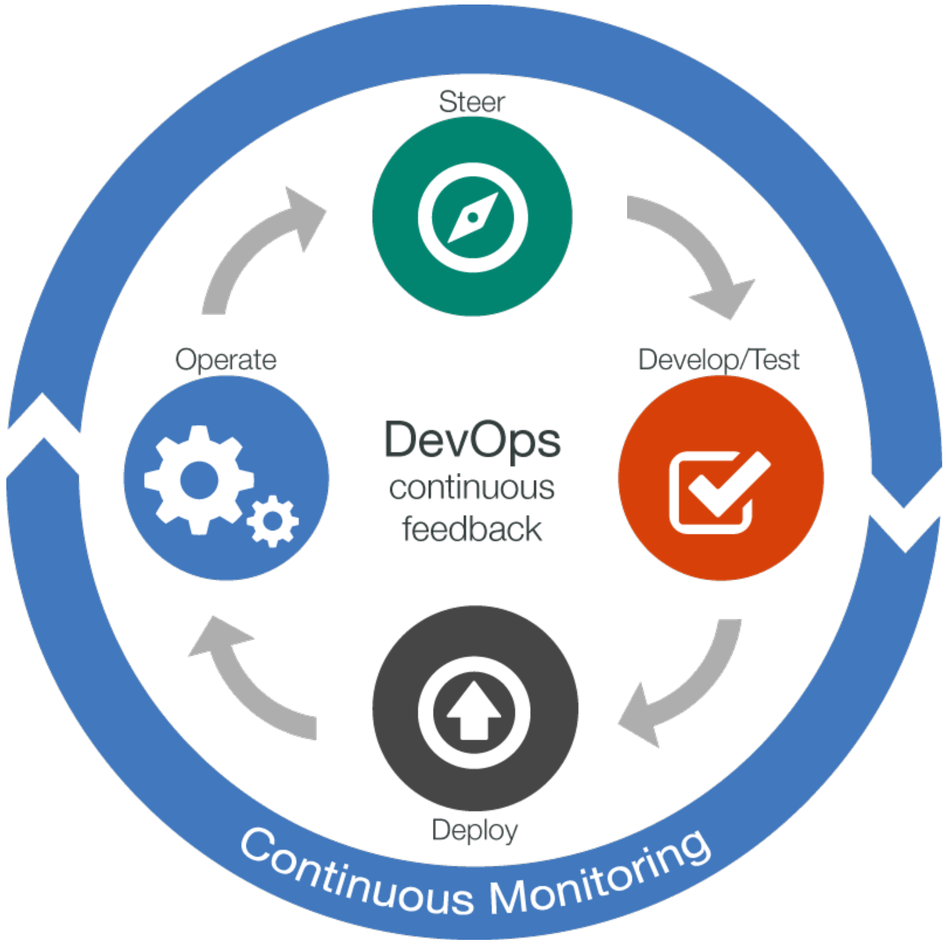
https://www.infoq.com/articles/devops-organizational-culture

“Don't fear failure. Fear being in the exact same place next year as you are today.”

Technology is the easy part



DevOps Summary



- A Cultural Movement
- Emphasizing Collaboration, Sharing, and Transparency
- Promoting Automation and Infrastructure as Code
- Achieving Continuous Integration and Delivery of Changes
- Immutable Delivery
- With One set of Metrics to rule them all

DevOps Maturity Matrix*

* Cloud Computing: The Cloud and DevOps by Dave Linthicum

Maturity Level	People	Process	Technology
Level 1 Ad Hoc	• Silo based • Blame and finger-pointing • Dependent on experts • Lack of accountability	• Manual processes • Tribal knowledge the norm • Unpredictable and reactive	• Manual builds and deployments • Manual testing • Environmental inconsistencies
Level 2 Repeatable	• Managed communications • Limited knowledge sharing	• Processes established within silos • No standards • Can repeat what is known, but can't react to unknowns	• Automated builds • Automated tests written as part of story development • Painful but repeatable releases
Level 3 Defined	• Collaboration exists • Shared decision making • Shared accountability	• Process automated across the software life cycle • Standards across organization	• Automated build and test cycle for every commit • Push button deployments • Automated user and acceptance testing
Level 4 Measured	• Collaboration based on shared metrics with a focus on removing bottlenecks	• Proactive monitoring • Metrics collected and analyzed against business goals • Visibility and predictability	• Build metrics visible and acted on • Orchestrated deployments with automatic rollbacks • Nonfunctional requirements defined and measured
Level 5 Optimized	• A culture of continuous improvement permeates through the organization	• Self-service automation • Risk and cost optimization • High degree of experimentation	• Zero downtime deployments • Immutable infrastructure • Actively enforce resiliency by forcing failures

Key Takeaways

- DevOps is about breaking down the silos and working as a Single Agile Team
- Culture is the #1 success factor in DevOps. Building a culture of shared responsibility, transparency and faster feedback is the foundation of every high performing DevOps team
- DevOps starts with learning how to work differently. It embraces cross-functional teams with openness, transparency, and respect as pillars
- Being able to recover quickly from failure is more important than having failures less often
- Measurements should encourage innovation and collaboration, and not punish failure (blameless culture)

“If you want to build a ship, don’t drum up the men to gather wood, divide the work, and give orders.
Instead, teach them to yearn for the vast and endless sea.”

– Antoine de Saint-Exupery*

* Author of The Little Prince

We don’t "DO" DevOps

We become DevOps