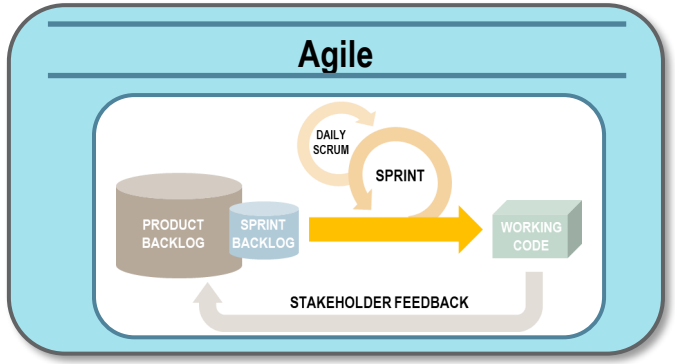
Agile/MOSA/Microservice Architectures Stronger Together

Kurt Mohr & Darryl Nelson
RI&S Engineering

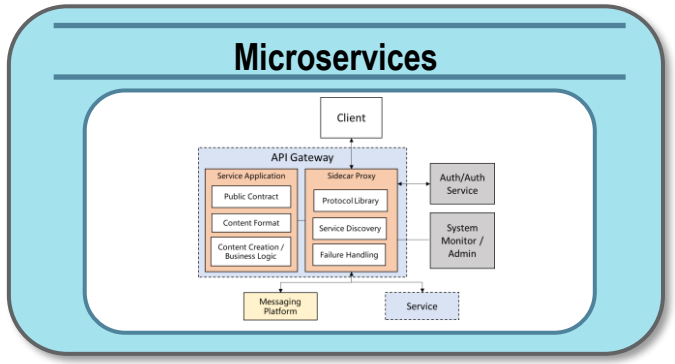
November 2, 2022

Agenda

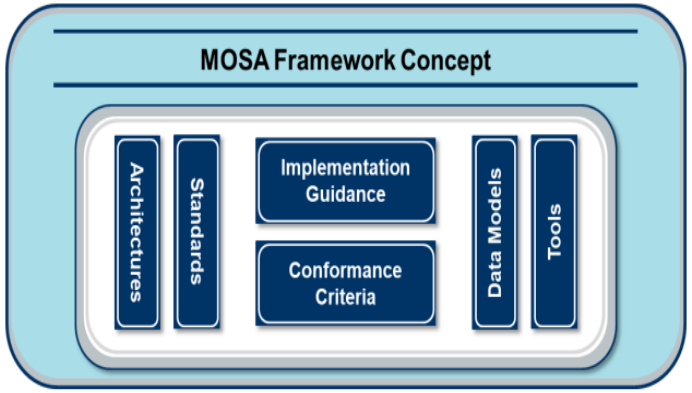
- **Establish a Lexicon** – defining Agile, Microservices, and MOSA
- **Speed of Agility** - agility and speed are keys to successful deployment of incremental capabilities
- **Microservice interfaces** - microservice architectures can represent both major and minor interfaces
- **Interface Stability of MOSA** - MOSA, with more stringent interface adherence at the major level, may be viewed as conflicting with rapid changes and trial and error of Agile
- **Adaptability vs. Stability** - using microservices to develop rapidly adaptable systems in an agile manner while meeting the MOSA requirements of modular design at the major system level, using consensus -based standards, and having severability at all levels of the architecture
- **Architectural Strategies** - architectural strategies for internal and external interfaces, and interface compatibility/extensibility
- **Reference Architecture** – what does 'good' look like
- **Commercial Examples** - interface stability at the major system level while providing adaptability and modularity at the lower levels of system architectures
- **DevSecOps** - enable speed and agility of microservice architectures while ensuring the quality of service required for MOSA system architectures



+



+



https://ac.cto.mil/wp-content/uploads/2020/06/MOSA-Ref-Frame-May2020.pdf

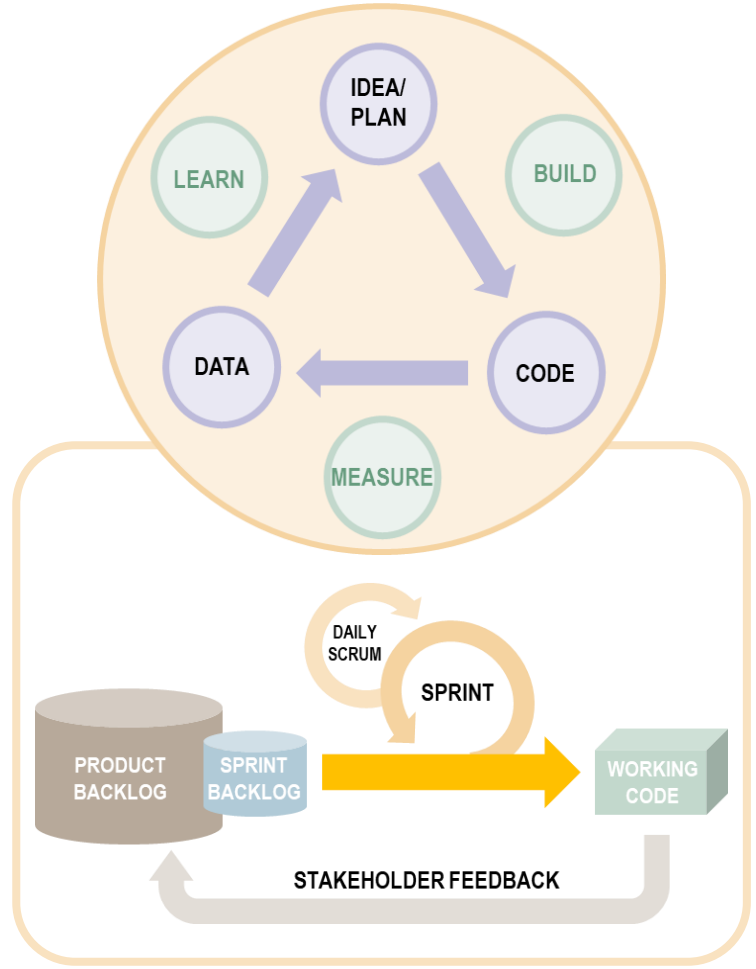
Lexicon - Agile

- Many Agile methodologies, but same basic values:
  - **Individuals and interactions** over processes and tools
  - **Working software** over comprehensive documentation
  - **Customer collaboration** over contract negotiation
  - **Responding to change** over following a plan

agilemanifesto.org/

- Focus on **Agility**
  - Ability to move or adapt quickly
  - Better Software through tight Feedback Loops for Rapid Adaptation
  - Facilitates fast learning

- Allows **rapid change** based on new information and discovery
  - Learning lessons and trying new approaches (retrospectives)
  - Buying down risk through experimentation (prototyping)

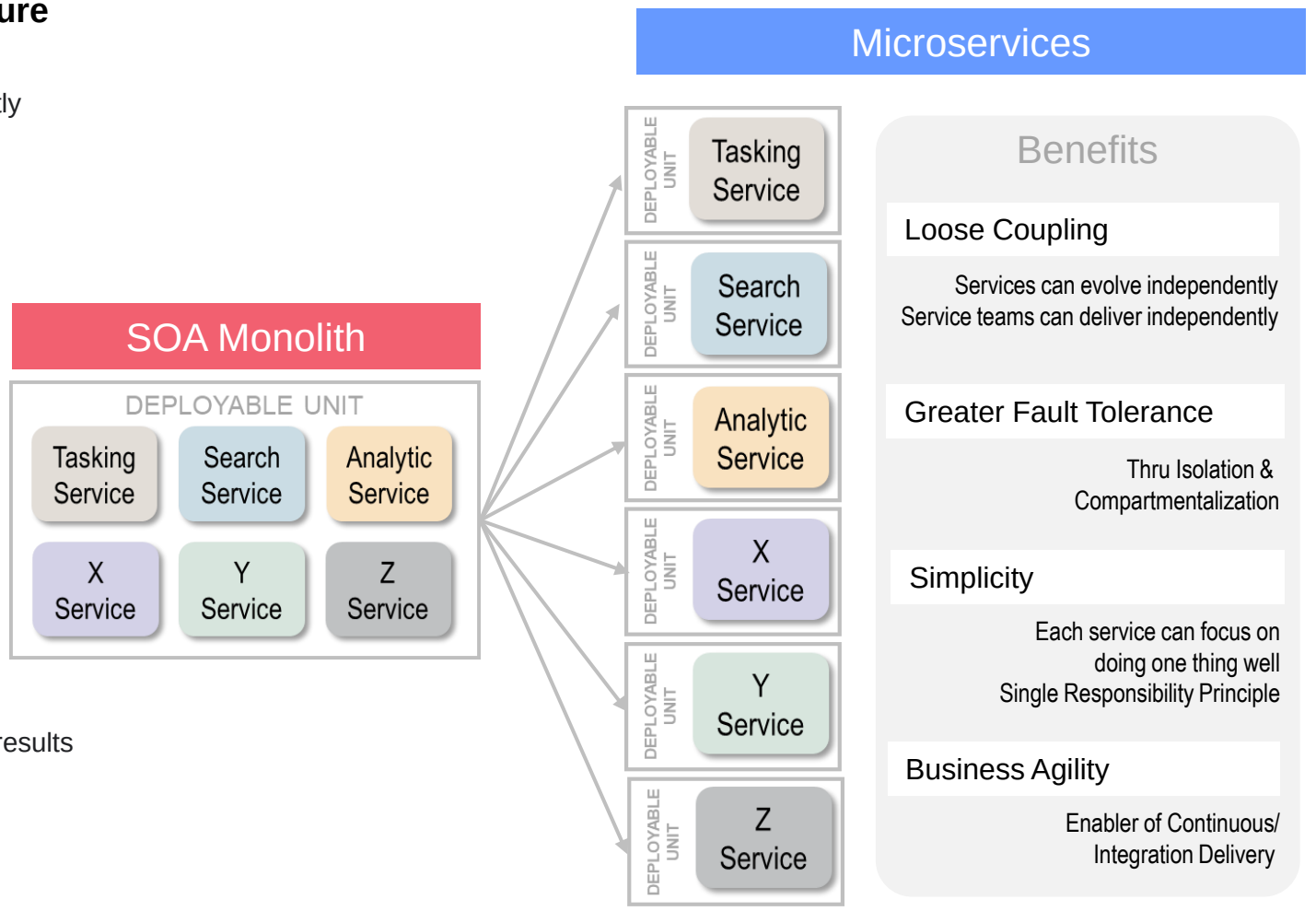


Agility is about adapting rapidly and not being tied to an original plan

Lexicon – Microservice Architecture

- **Next evolution of Service-Oriented Architecture (SOA) to Microservices**
  - Suites of software applications composed of independently deployable services
  - Loosely coupled services
  - Fine-grained, meaning small in size and scope
  - Light-weight protocols
  - Independently managed
  - Delivering systems as a set of constrained, granular, independent collaborating services

- **Value of Microservice Architectures**
  - Enforces Isolation for resilience and elasticity
  - Ability to create and deploy fixes independently
  - Limiting the reach of defects
  - Minimizes the need for specialized personnel knowledge
  - Can be assembled in different ways to produce different results
  - The independent services can be composed together to cooperate, providing sophisticated business services

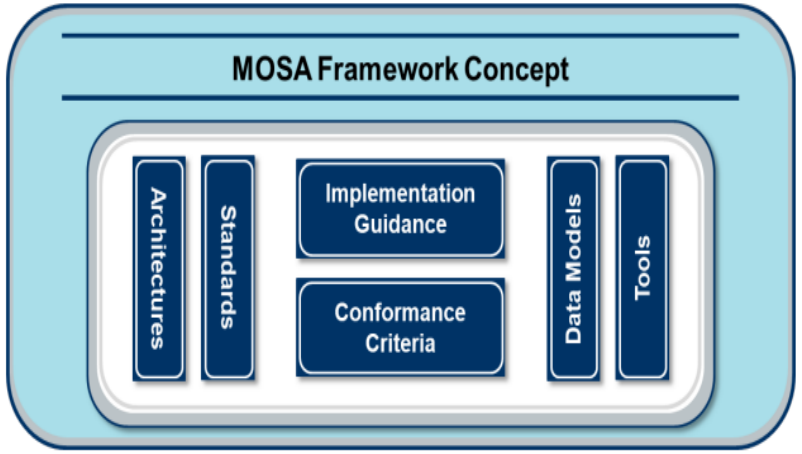


Microservice Architectures maximizes the number of 'services' to build on

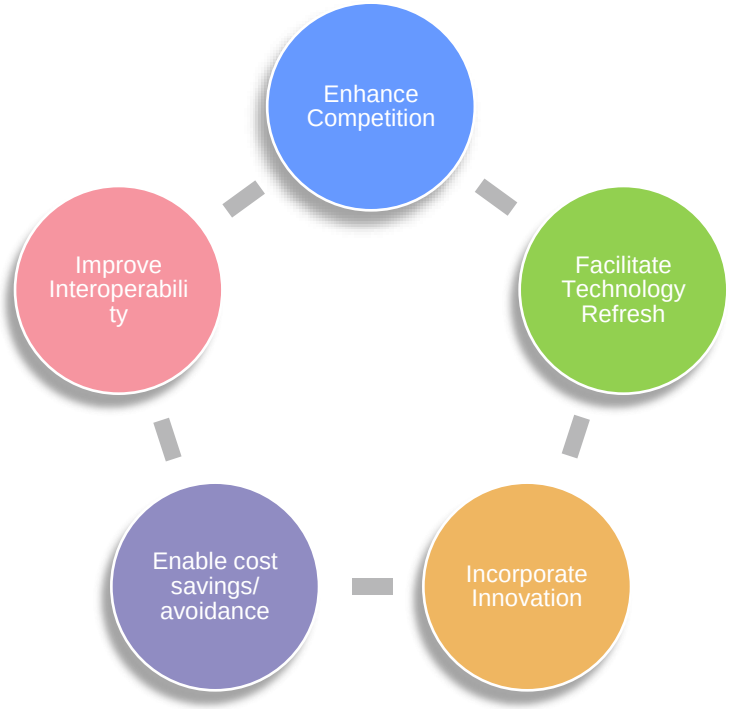
Lexicon - MOSA

- **MOSA Definition**
  - A modular design that uses major system interfaces between major platforms/systems/components
  - Ensures major system interfaces use widely supported and consensus-based standards
  - Allows major system components severability at the appropriate level to be added/removed/replaced
- **MOSA intent is to:**
  - Support a more rapid evolution of capabilities and technologies throughout the product life cycle
  - Ensure architecture modularity, open systems standards, and appropriate business practices.
- **Expectation is MOSA delivers:**
  - Enhance competition
  - Facilitate technology refresh
  - Incorporate innovation
  - Enable cost savings/cost avoidance
  - Improve interoperability

References
- https://www.dsp.dla.mil/Programs/MOSA/
- https://ac.cto.mil/mosa/



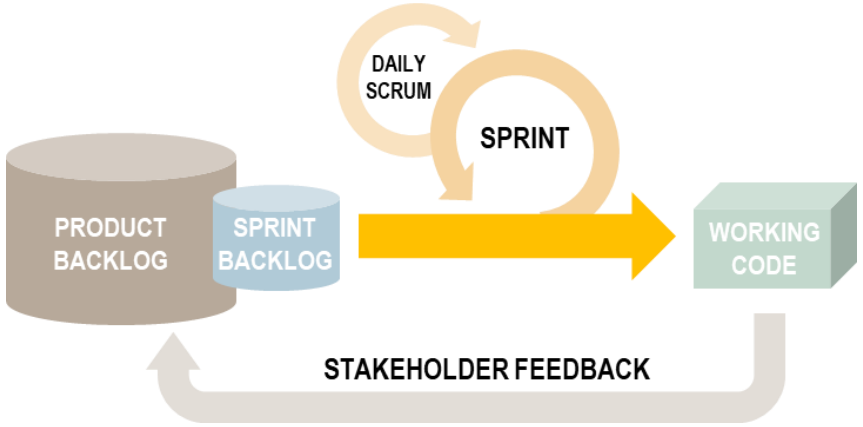
https://ac.cto.mil/wp-content/uploads/2020/06/MOSA-Ref-Frame-May2020.pdf

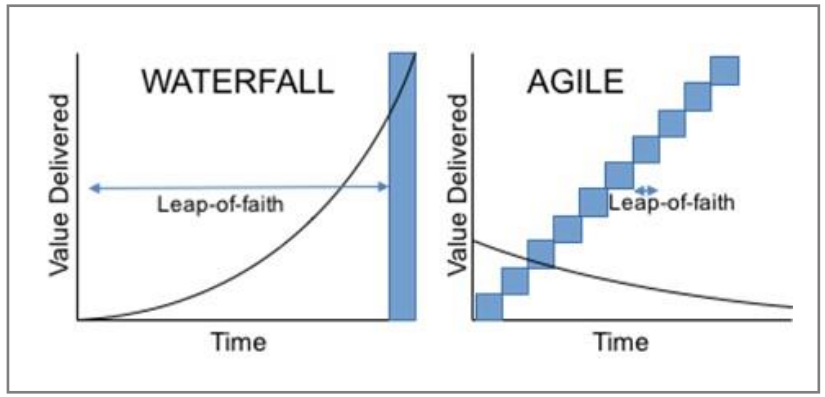


MOSA employs standards to allow component replacement

Speed of Agility

- **Agile methodologies deliver capabilities quickly at lower risk**
  - Incremental deliveries on a regular cadence
  - Short cycle times get things to users for evaluation
- **Experimentation allows new technologies to be evaluated**
  - "Fail Fast" methodology to determine what works and what doesn't
  - No long development cycles
- **Minimal Viable Product (MVP) can be found and refined**
  - Many times the "most important features" evolve
  - Eliminating unnecessary features can save cost and schedule
- **Testing early and often reduces tail end schedule risk**
  - "Test gets compressed" mentality changes to "shift left"

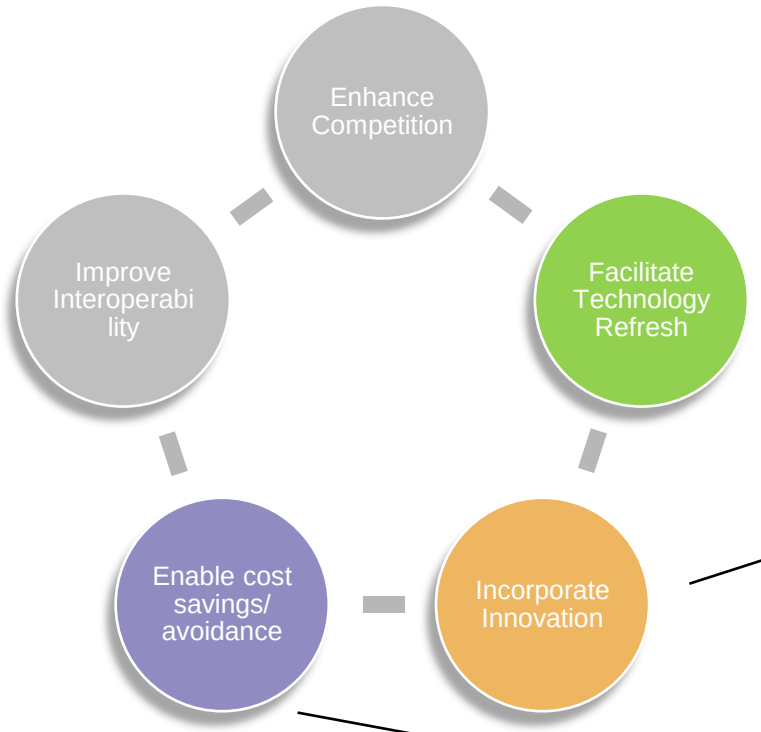




Dr Ian Mitchell, CC BY-SA 4.0 https://creativecommons.org/licenses/by-sa/4.0, via Wikimedia Commons https://commons.wikimedia.org/wiki/File:Waterfallvsagile.jpg

Agile shortens timelines to get things into customer's hands

Agility + MOSA – Contradictory or Complimentary?



Facilitate technology refresh

- Agility can shorten technology refresh cycles

Incorporate innovation

- Experimentation and team-driven priorities enables innovation

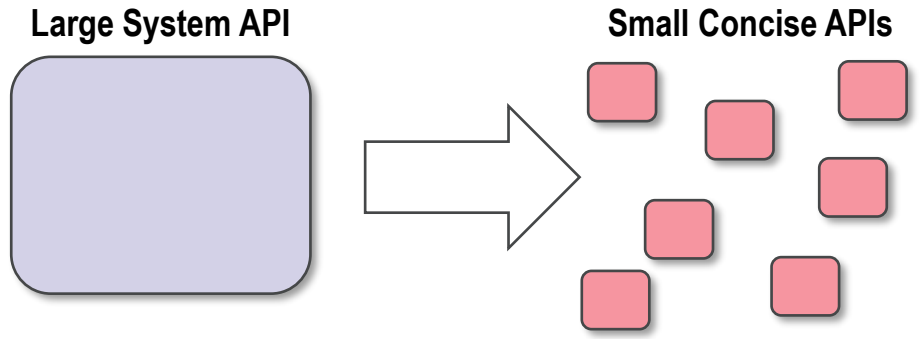
Enable cost savings/cost avoidance

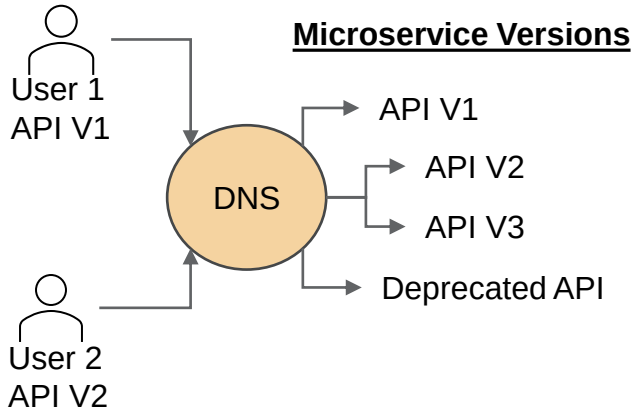
- Risk mitigation and rapid fielding support cost avoidance through early testing
- Customers can focus on MVP

Agility provides several of the outcomes MOSA seeks to achieve

Microservice Flexibility

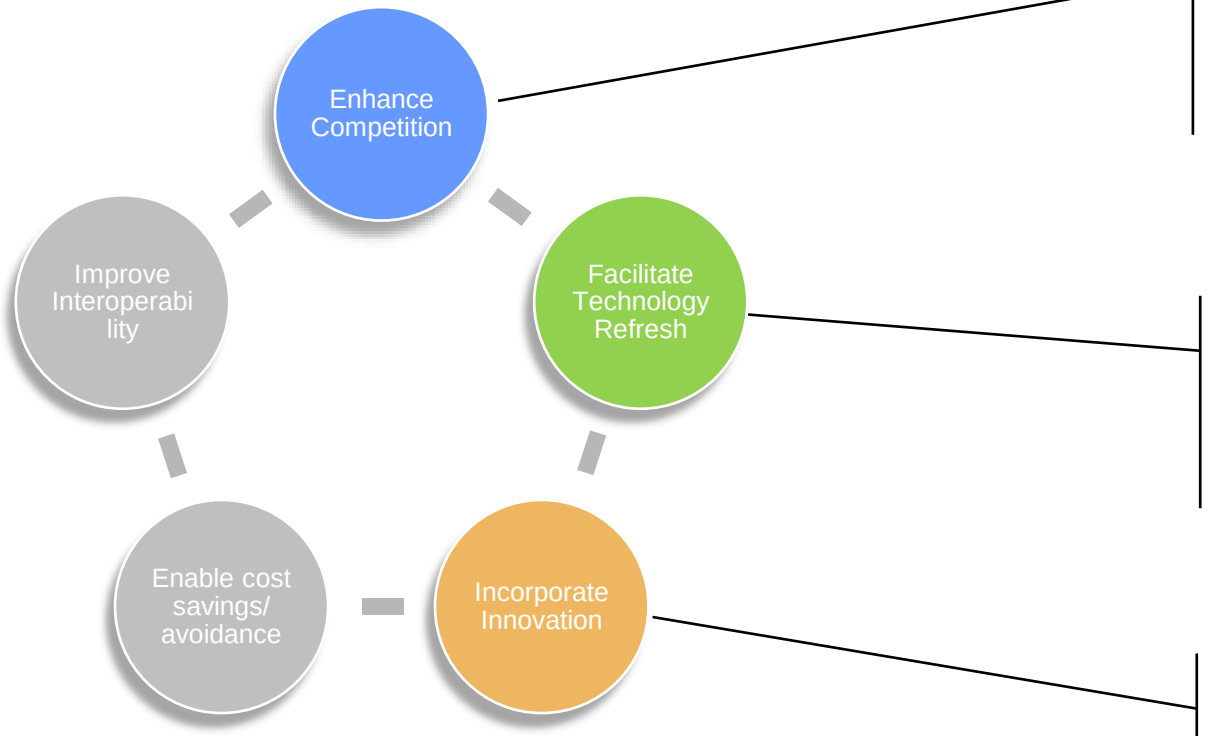
- **Microservices have individual interfaces**
  - Instead of one large interface, many smaller ones
  - Small, concise interfaces are easier to understand
- **Deployment can be targeted**
  - Deploying a single service for a test or a set for added functionality
- **Interface changes don't require all parties aligning schedules**
  - Interfaces can be versioned and operate concurrently
  - Deprecating old interfaces gives time for callers to adapt
- **Changing out a microservice provider is low impact**
  - Can operate the old and new concurrently
  - Changing the endpoint can be as easy as a DNS entry





Microservice architectures decouple implementation from interface

Microservices + MOSA – Can they live together?



Enhance competition

- Microservices allow competing services to be evaluated
- Services can live side by side for comparison or use

Facilitate technology refresh

- New versions can co-exist with older ones for refresh
- Microservices can be updated independently

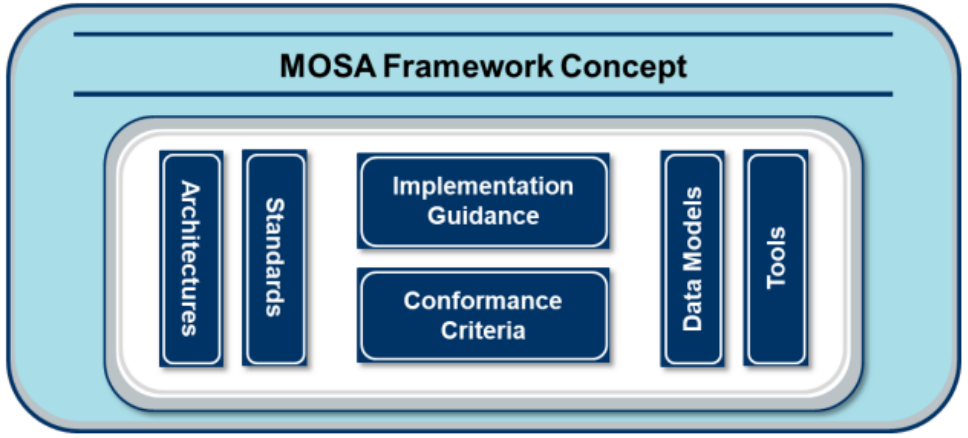
Incorporate innovation

- Services enable deployment of MVP with rapid iteration

Microservices have the potential to aide MOSA objectives

MOSA Interface Stability

- **But MOSA is looking for interface stability, right?**
  - A modular design that uses **major system interfaces** between major platforms/systems/components
  - Ensures major system interfaces use **widely supported and consensus-based standards**
  - Allows **major system components severability** at the appropriate level to be added/removed/replaced
- **MOSA focuses on the major systems**
  - MOSA is silent on the minor systems and components
  - There is no clear definition of “major”
- **Interfaces are tightly controlled and versioned**
  - Allowing ad-hoc updates would be chaotic at best
  - Version changes go through multiple reviews with many stakeholders



Modular Open Systems Approach (MOSA) Reference Frameworks in Defense Acquisition Programs, Deputy Director for Engineering, Office of the Under Secretary of Defense for Research and Engineering, Director of Defense Research and Engineering for Advanced Capabilities, May 2020, Distribution Statement A. Approved for public release. Distribution is unlimited.
https://ac.cto.mil/wp-content/uploads/2020/06/MOSA-Ref-Frame-May2020.pdf

MOSA’s key concerns are major interface boundaries

Adaptability vs. Stability

Adaptability

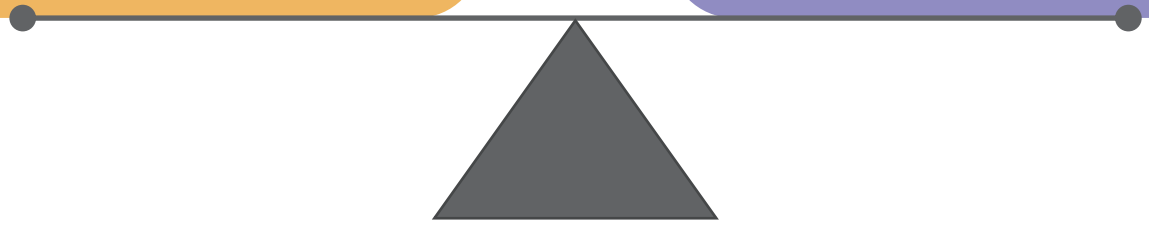
Agile and Microservices provide **adaptability**

- Both in implementation details and interface changes
- Adding new services or updating them rapidly
- Running concurrently or deviating for specific needs
- Accepting trial and error to 'learn' the best approach
- Technology change at the most granular level
- Every microservice is replaceable

Stability

MOSA provides interface **stability**

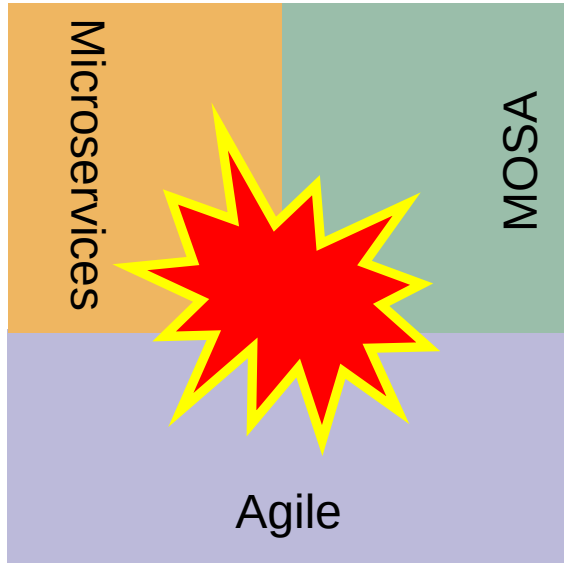
- Rarely changing interfaces
- Generally, one implementation in the system at a time
- Technology refresh of large system components
- Competition at the macro level



Ease of change vs. the stability for long contract cycles seem to be at odds

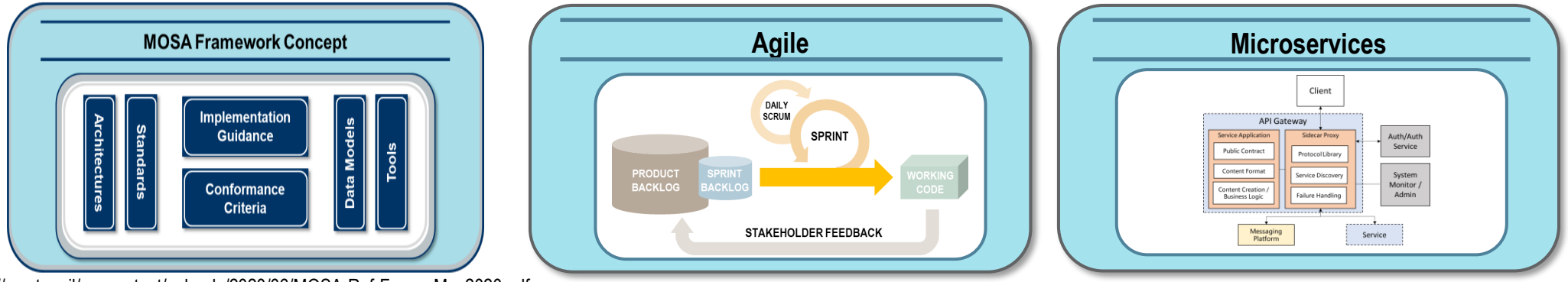
So Do they Conflict?

- **Microservices allows for frequent interface changes**
  - Agile allows for interfaces to be updated regularly
  - MOSA does not allow for interfaces to change in weekly Agile cycles
- **Agile experimentation isn't consensus-based**
  - MOSA relies on consensus for interfaces
  - Microservice changes could be 'tried' in an agile increment with a subset of callers
- **MOSA is serving a wide range of stakeholders**
  - Agile has stakeholders, but it does not require consensus
  - Microservices could have different versions for different stakeholders



So Agile and Microservices work, but not with MOSA?

Same Objectives, Different Approach

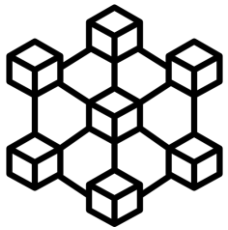


https://ac.cto.mil/wp-content/uploads/2020/06/MOSA-Ref-Frame-May2020.pdf

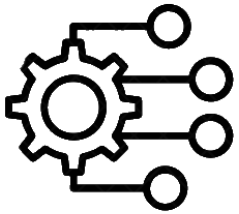
- **While the approaches differ, there are significant synergies**
  - Changes at the micro-level don't need to be coordinated
  - Microservices can be callers of major interfaces
- **Competition enhancement at all levels**
  - MOSA brings it at major components
  - Microservice architectures compete deeper in the architecture
  - "Competition" can be capability, not just provider, using Agile
- **Interoperability can be enhanced**
  - Rapid microservice adapters can insulate from major changes through design patterns
    - Façade pattern, Delegation pattern, and Proxy pattern
  - Interface versioning can be implemented quickly in Agile

Combining Agile, MOSA, and Microservices are Different but Complementary Approaches
They Serve the Same Objectives

Architectural Strategies



- **Defining Major and Minor Interfaces**
  - Architecturally defining which interfaces are major and thus MOSA
  - Providing API registration for interface selection of minor interfaces
  - Using extensible interface protocols
    - E.g. Adding optional content to JSON messages

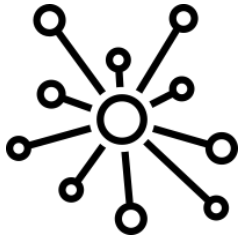


- **Intelligent Routing**
  - Routing data based on the recipient's needs
  - Changing routing rules to old or new service implementations
  - Identifying interfaces that are backwards compatible or where multiple versions could co-exist

Multiple strategies can be used to marry the approaches

Architectural Strategies cont.

- **Data Adaptation**
  - A microservice can be inserted to adapt new interface calls to old
    - E.g. Stripping new fields so that a recipient can continue to function
  - Developed, tested, and deployed quickly through Agile
  - When the recipient is ready for the new interface, the microservice is disabled or updated

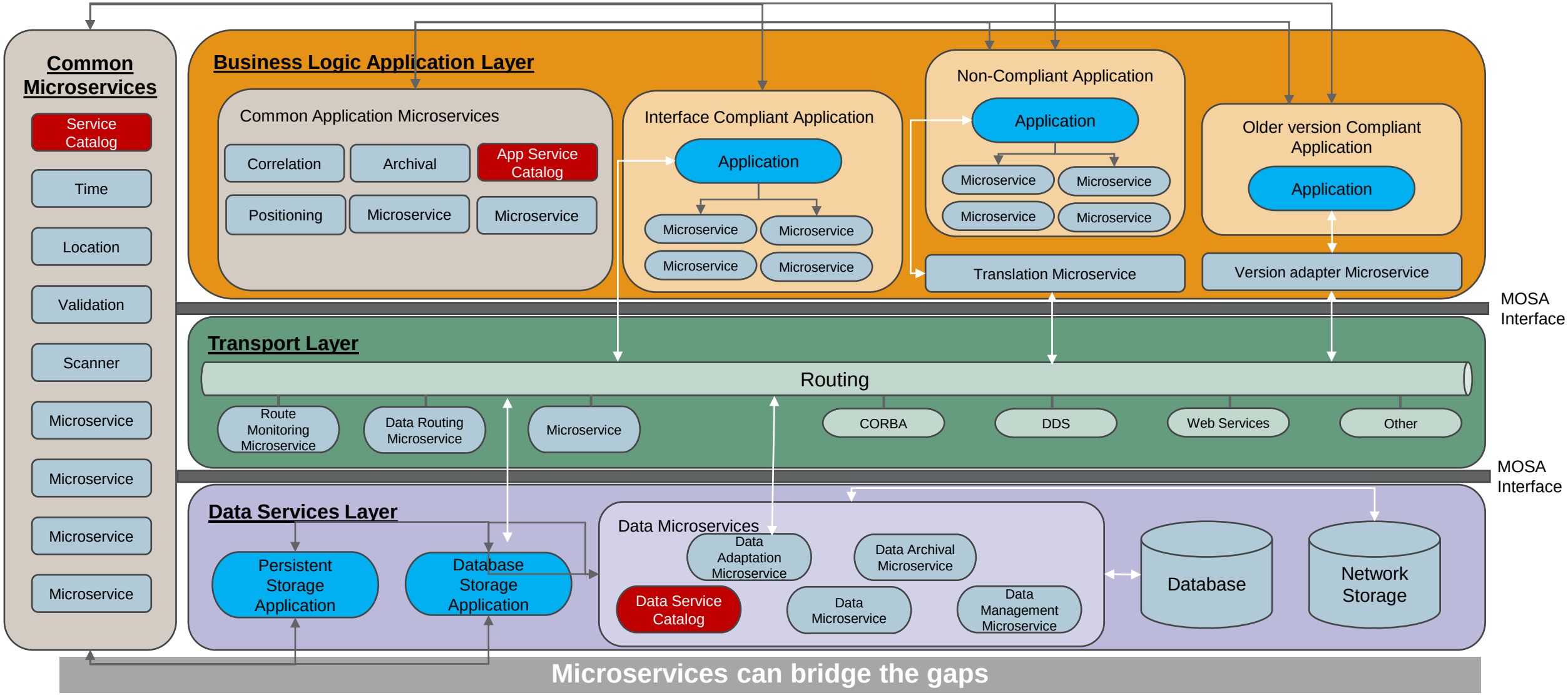


- **Maximizing Stateless Processing**
  - Decoupling data state between components to reduce major interface dependency
  - Using proxy interfaces to deal with ‘extra’ data across major interface points

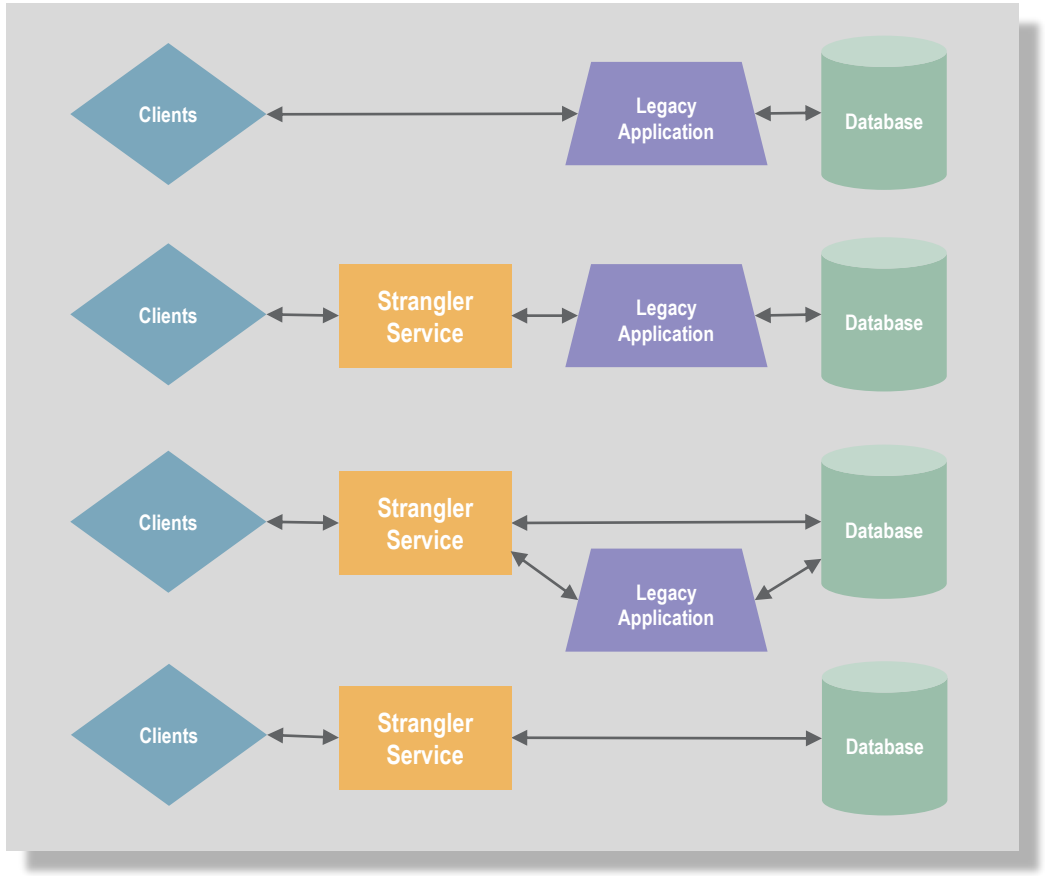
- **Using Big Data Approaches**
  - Segregating responsibilities to the lowest levels
  - Using data aggregation services to initiate major interface calls

MOSA objectives are all about dependencies, minimize those

Reference Architecture



Strangler Pattern: Incremental Modernization

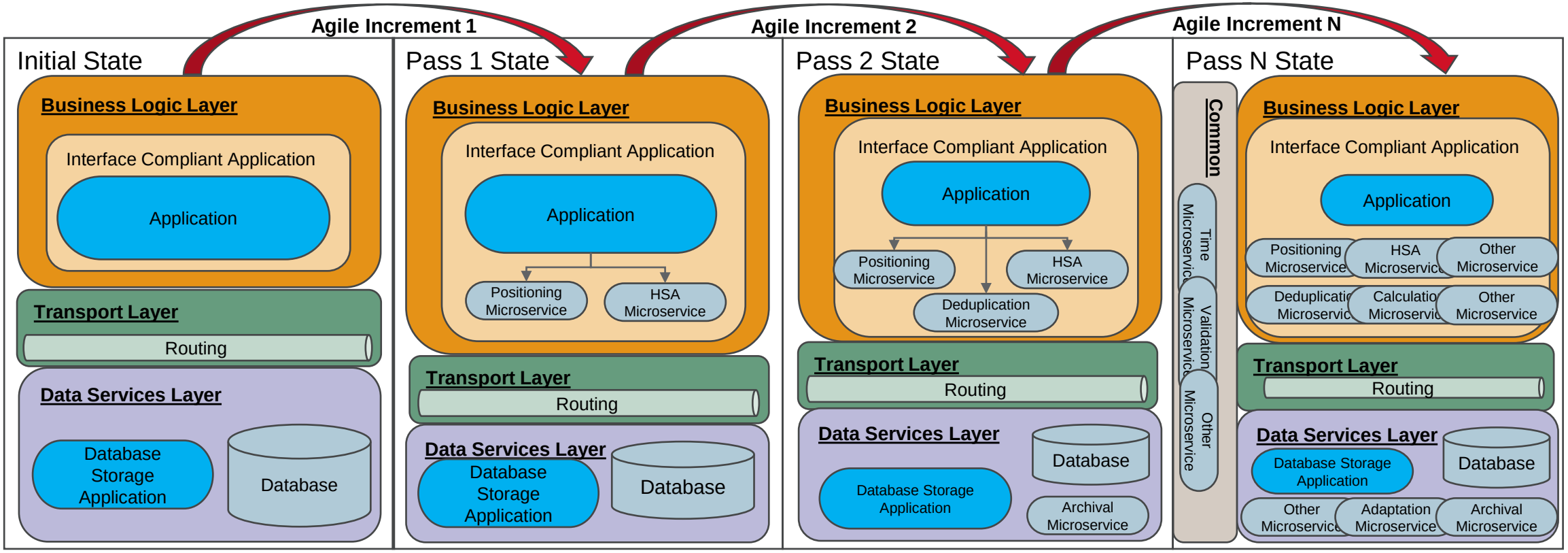


1. Original legacy application
2. Strangler as a pass through
3. Strangler taking over some responsibilities
4. Complete transition to Strangler

A *software pattern* of incremental migration of a legacy system by replacing existing functionalities with new applications and services in a phased approach.

Agile Migration Paths

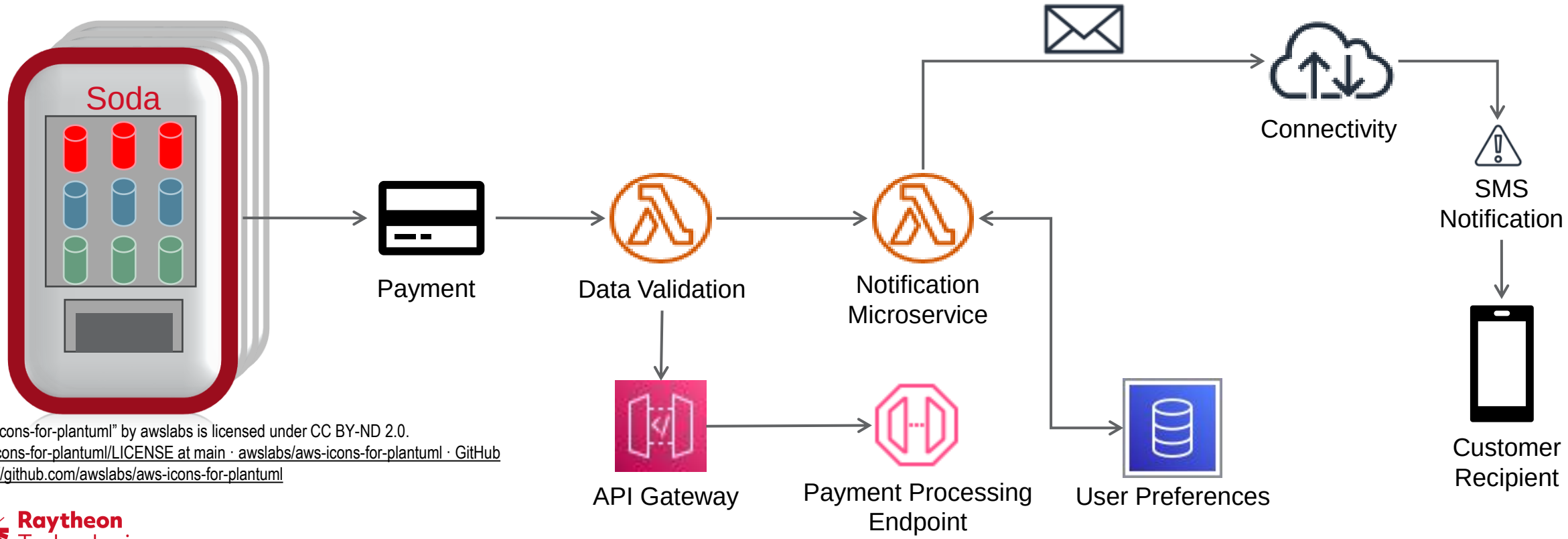
- Use Strangler pattern for incremental modernization to an application by adding microservices
- Agile methods provide boundaries to limit risk each increment



Migration doesn't have to be a "big bang"

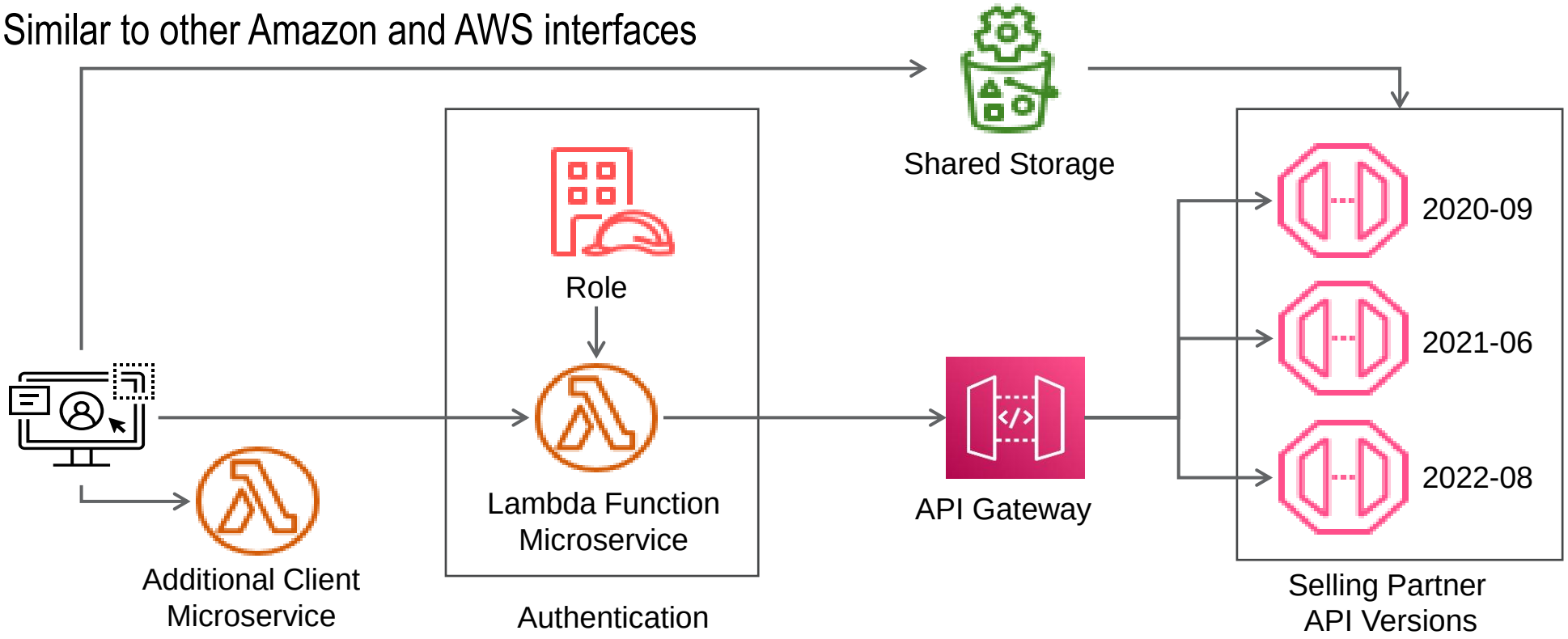
Commercial Example

- **Soda manufacturer vending machines**
  - Vending machines call microservice
    - Implemented as Lambda function
  - Data is conformed to major payment APIs
  - Newer vending machines updated to use latest interface features
  - Microservices send notifications to users



Commercial Example

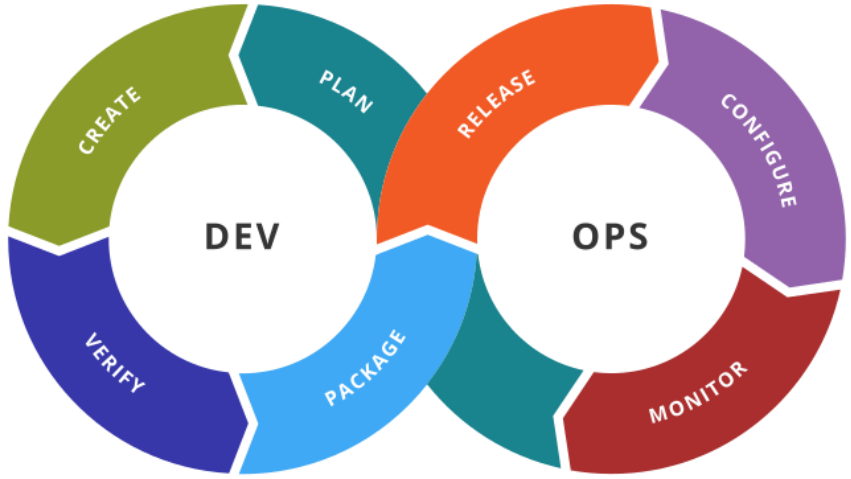
- **Amazon Selling Partner interface**
  - Supports multiple versions of agreed to static interfaces
  - Concurrent execution routed by URL
  - Similar to other Amazon and AWS interfaces
  - Data adaptation and role restrictions
  - Callers can easily abstract their version away



What about DevSecOps?

DevSecOps allows rapid deployment with reduced risk posture

- Deploys micro-services at the lower levels while maintaining interface code stability
- Reduces risk of full system deployments
- Significantly reduces or eliminates downtime for deployments



Kharnagy, CC BY-SA 4.0 https://creativecommons.org/licenses/by-sa/4.0, via Wikimedia Commons
https://upload.wikimedia.org/wikipedia/commons/0/05/Devops-toolchain.svg

Improves security posture

- Allows targeted patching of critical security changes
- Enables automated security testing on granular level

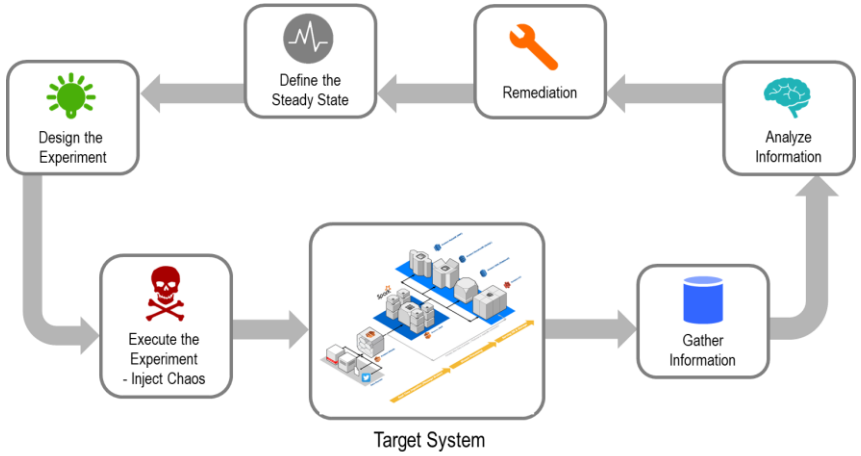
Automated verification finds issues early

- Validates that the interfaces didn't change
- Regression tests the microservice functionality
- Allows validation of multiple versions concurrently in operations

DevSecOps supports MOSA, Agile, and Microservice approaches

Microservice Recommendations

- ***UX is Not Just for UI's***: Design the interface contract FIRST with stakeholders
  - Tools: e.g. Swagger
- **Patterns**: Use them!
  - Recommended book: https://www.manning.com/books/soa-patterns
- Use **Frameworks**
  - e.g. Spring
- **No one tool is right for every job**
- Adopt and Integrate ***Chaos Engineering***
  - *Chaos Engineering is the discipline of experimenting on a distributed system in order to build confidence in the system's capability to withstand turbulent conditions in production.*[1]



[1] https://principlesofchaos.org/

Thank you.

Authors

Kurt Mohr

Kurt Mohr is an Engineering Fellow with over 20 years in Software Engineering and Architecture spanning both commercial and defense. He has a strong background in multiple Agile development methodologies including Scrum, Kanban, and Scaled Agile. Kurt is the Software Technical Director for Raytheon Intelligence & Space Common Engineering. Kurt has architectural experience with all forms of software systems including cloud, bare metal, and embedded as well as highly scalable distributed computing.

Darryl Nelson

Darryl Nelson is the Technology Director for *Data Engineering and Advanced Software* at Raytheon Intelligence & Space. Darryl's previous roles include Chief Engineer, Corporate Technology Area Director, and Chief Scientist. He currently specializes in scalable system architectures with a focus on operationalizing AI, distributed data management, and continuously evolving software systems. Darryl is a US Army veteran with an interest in the impact of software on the future of warfare.