RISK AND RESILIENCE BOOTCAMP

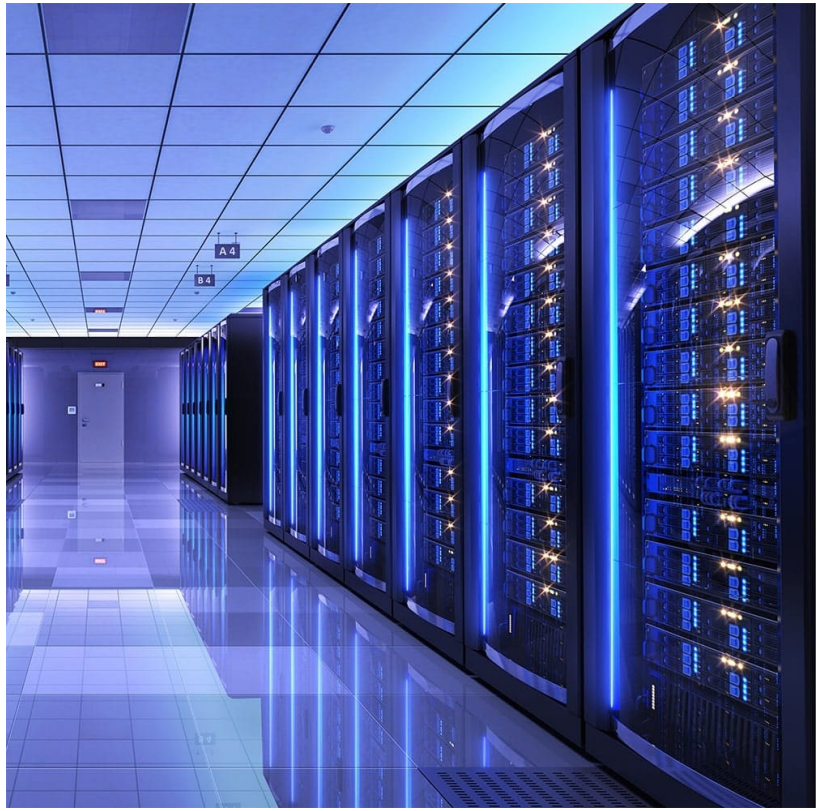
TEKsystems Global Services

WORKFORCE DEVELOPMENT

CORE TECHNOLOGY OPERATIONS

In this module, we will

- Examine IT infrastructure, technology and processes from a risk and resiliency perspective
- Review how risk management and systems resilience deal with the challenges of technology operations
- Review the role of process controls and standard operation procedures



INTRODUCTION

- The IT infrastructure and associated processes and technology tend to be sprawling and complex in most organizations
  - This section will look at this infrastructure from several different perspectives
- This first section will include a glossary of terms
  - This will help clarify the discussions that follow
  - These are not intended to be definitive or universal definitions

IT INFRASTRUCTURE

- The collection of physical and virtual resources that support IT services and operations
  - Includes servers, storage, networks, databases, middleware, and cloud resources
  - Any other hardware or software asset used in IT operations
- Potential risk/resilience issues
  - Legacy infrastructure is a source resilience risks due to age, vendor support limitations, or hardware dependencies
    - Includes existing hardware, mainframe systems, older commercial and in-house applications and software
  - Cloud and virtualized infrastructure introduces different categories of risk factors
    - For example: shared responsibility, vendor lock-in, and reliance on third-party SLAs
  - Vulnerabilities in infrastructure require preventive and detective controls
    - For example: patching gaps, misconfiguration, dependencies on obsolete components or software

IT ARCHITECTURE

- The structured design and blueprint of IT systems
  - Defines how infrastructure components, applications, data flows, and processes interact to support business goals
- Potential risk/resilience issues
  - Architectural decisions (e.g, monolithic vs. microservices) influence both operational risk and resiliency strategies
  - Poorly documented or outdated architectures in legacy systems increase the likelihood of cascading failures and can hinder recovery plans and efforts
  - Risk mapping exercises often begin by analyzing the IT architecture to identify single points of failure, hidden dependencies, and integration risks

APPLICATION PORTFOLIO

- Applications are the business-facing layer of IT
  - They “sit on top” of infrastructure like servers, databases, networks, cloud resources
  - They are shaped by software architecture
    - For example: monolithic applications, microservices, client server systems, etc
  - Applications inherit risks from infrastructure and architecture and introduce other risks
    - *Code quality issues:* bugs, security vulnerabilities
    - *Integration complexity:* risk of cascading failures
    - *Update and patch cycles:* downtime, testing risk
    - *Business dependency:* operational or reputational risk if services are unavailable
  - Summary
    - Infrastructure = what the applications run on
    - Architecture = how the applications are structured and connected
    - Applications = the tools and services that directly deliver business value

GLOSSARY TERMS

- Legacy Systems
  - Older technologies, applications, or hardware that remain in production despite being outdated or obsolete
  - Legacy systems are hard to patch, costly to maintain, and often rely on increasingly scarce expertise, making them high-risk in both security and resiliency
    - *"The payroll system is a legacy platform that requires manual intervention, creating an operational risk"*
- Standard Operating Procedure (SOP)
  - Documented set of instructions detailing how to perform routine operations
  - SOPs reduce variability, ensure consistent controls, and support recovery under stress
  - Updated regularly, SOPs document the current optimal way to execute an operation
  - SOPs represent a knowledge base of the best practices used in the organization

GLOSSARY TERMS

- Preventive Controls
  - Measures put in place to reduce the likelihood of risk events before they occur.
    - For example: redundancy, patch management, access controls
    - *"Implementing multi-factor authentication is a preventive control against credential theft"*
- Detective Controls
  - Measures designed to identify and alert on risk events once they occur
    - For example: monitoring dashboards, SIEM alerts, intrusion detection
    - SIEM = Security Information and Event Management
    - *"Log monitoring served as a detective control to catch anomalous activity"*

GLOSSARY TERMS

- Corrective Controls
  - Measures aimed at restoring systems to normal operation and minimizing damage after an incident
    - For example: backups, failover systems, hot site recovery
    - Often tested using testing scenarios like disaster recovery drills
- Operations Monitoring
  - Tools and processes for observing performance, availability, and security of IT operations in real time
  - Monitoring is critical for both detective and preventive risk management
    - For example: AWS CloudWatch provides monitoring of workloads for risk indicators

GLOSSARY TERMS

- Risk Mapping (Operations)
  - Identifying and charting risks across infrastructure and processes
  - Often using various diagrams and other analytic tools
  - Identifies lack of necessary controls for mitigation
    - For example: A risk map might highlight backup failures as a critical operational gap
- Recovery Models
  - Structured approaches to restoring operations after an outage or disaster
  - Hot site, cold site, active-active clusters, cloud-based disaster recovery
    - Active-active clusters, for example, require two copies of the production system running at all times
    - An active-active recovery model provides continuous availability but is costlier

GLOSSARY TERMS

- Business Resilience Lifecycle (BR Lifecycle)
  - The continuous cycle of planning, testing, executing, and improving resiliency strategies for business operations
  - Covers preparedness, incident response, recovery, and lessons learned
  - The BR lifecycle ensures lessons from incidents are fed back into preventive controls

IT SCOPES OR VIEWS

- Development
  - The creation or customization of application software
  - In automated CI/CD environments, this also includes development of tooling
    - Tooling is software used to build other software, like code repositories, automated piplines
  - In DevOps environments, this also refers writing code to deploy virtual infrastructure
- Operations
  - Refers to deployment, monitoring and managing of running software
  - Also includes the management of hardware and other infrastructure

MISSION CRITICAL SYSTEMS

- Derived from the idea of mission critical software
  - Refers to the phenomenon of organizations becoming so dependent on software in their day-to-day operations that they cannot function if the software fails
  - First proposed by Grady Booch
- The software runs on the IT infrastructure
  - Means that failures in the operations environment can also cripple the organization
  - Organizations cannot function without a fully operational IT infrastructure

INDUSTRIAL STRENGTH IT

- Booch pointed out that as software becomes mission critical
  - It starts to permeate the whole organization
  - As a result, it expands in size and functionality until it becomes what he calls *"Industrial Strength Software"*
- Industrial strength software means
  - The code base is so large that no one individual can understand it
  - Often includes a lot of legacy software and systems, and their interdependencies
- Applying this to operations
  - The running IT systems become so complex, no one individual can understand the full scope of the operational environment

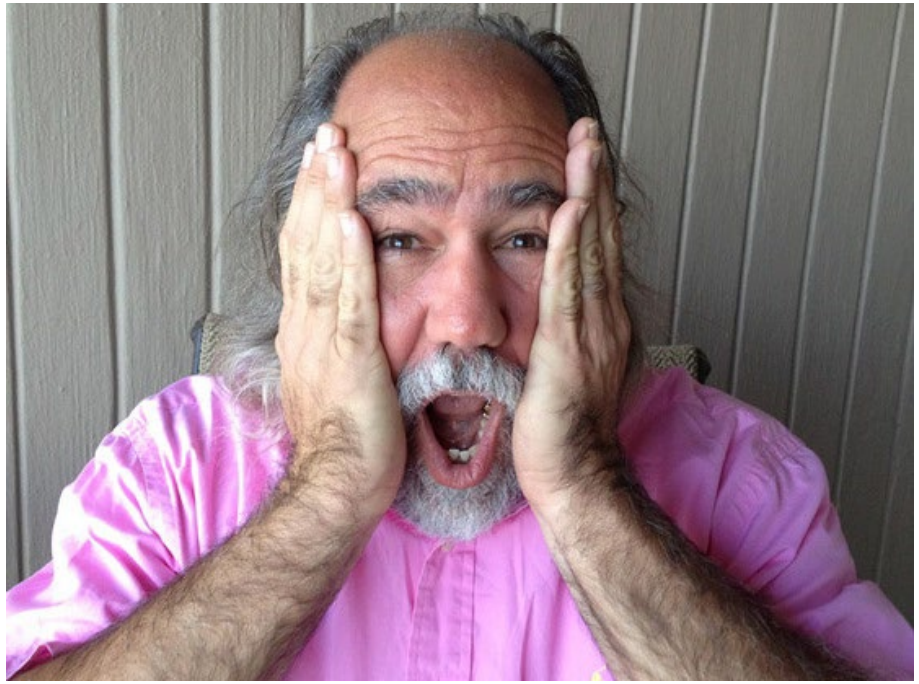
INDUSTRIAL STRENGTH IT

Mission critical software tends to have a long lifespan, and over time, many users come to depend on their proper functioning. In fact, the organization becomes so dependent on the software that it can no longer function in its absence. At this point, we can say the software has become industrial-strength.

The distinguishing characteristic of industrial strength software is that it is intensely difficult, if not impossible, for the individual developer to comprehend all of the subtleties of its design.

Stated in blunt terms, the complexity of such systems exceeds the human intellectual capacity. Alas, this complexity we speak of seems to be an essential property of all large software sys-tems. By 'essential' we mean that we may master this complexity, but we can never make it go away.

Grady Booch



THE COMPLEXITY PROBLEM

- As software and systems become industrial strength
  - They become more complex as they increase in scale
  - The increase in complexity is often exponential
  - Both from the codebase perspective and the operational perspective
- Why this matters
  - The more complex a system is the more brittle it can become
  - The more complex a system is, the more prone it is to failures of various types
  - Complexity exposes a larger risk surface
  - Complexity makes it difficult to determine where to apply remediation
  - Complexity also makes it more difficult to recover from outages
    - Often because the source of the failure isn't obvious

THE COMPLEXITY PROBLEM

The United States is losing almost as much money per year to IT failure as it did to the [2008] financial meltdown. However the financial meltdown was presumably a onetime affair. The cost of IT failure is paid year after year, with no end in sight. These numbers are bad enough, but the news gets worse. According to the 2009 US Budget [02], the failure rate is increasing at the rate of around 15% per year.

Is there a primary cause of these IT failures? If so, what is it?.... The almost certain culprit is complexity.... Complexity seems to track nicely to system failure.

Once we understand how complex some of our systems are, we understand why they have such high failure rates.

We are not good at designing highly complex systems. That is the bad news. But we are very good at architecting simple systems. So all we need is a process for making the systems simple in the first place.

Roger Sessions

SCALING IN SYSTEMS

- Scaling is an increase in size or quantity along some dimension
- Can take place in the development or operations space
- Development scaling
  - Increase in complexity, functionality or volume of code
    - These dimensions are often related
  - Business analogy is a company increasing the range of services and products they offer or expanding into different markets (like a Canadian company expanding into Europe)
- Operational scaling
  - Increase in the amount of activity of a system
    - For example: throughput, load, transaction time, simultaneous users, etc
  - Traditional architectures tend to be scalable only to a limited degree
  - There is a certain level of complexity after which they become unmaintainable

MANAGING COMPLEXITY

- Most of the topics in this section deal with risk and resiliency management in these large complex environments
- We will break this down into the scopes we looked at earlier
  - Divide and conquer: development, operations, usage and governance modules
  - And then integrate them into a cohesive whole at the end
- This module introduces two sample analytic tools
  - *The Zachman Framework:* A tool for modeling IT infrastructure and architecture
  - *Application Lifecycle Management*: An approach to modeling the lifecycle of an individual application

ZACHMAN FRAMEWORK

- Analytic tool used to understand complex business or IT systems.
- Uses two-dimensional grid
  - The columns represent basic questions of: *What, How, When, Who, Where*, and *Why*
  - The rows show how these questions are realized in the architecture
- Each column represents a basic question about the enterprise:
  - *What (Things):* What things are important? (e.g., data, information, objects)
  - *How (Function):* How do processes work? (e.g., functions, algorithms, workflows)
  - *Where (Network):* Where are things located? (e.g., nodes, connectivity, geography)
  - *Who (People):* Who is responsible? (e.g., roles, org structure, accountability)
  - *When (Time):* When do events happen? (e.g., schedules, timing, cycles)
  - *Why (Motivation):* Why are we doing this? (e.g., goals, rules, strategy)

ZACHMAN FRAMEWORK

- Each row answers the column's question from a different stakeholder's view, moving from abstract to concrete
  - Each row reifies or makes more concrete the row above it
- The rows are:
  - *Scope (Planner's View):* High-level context (e.g., business goals, external environment)
  - *Business (Owner's View):* Business perspective (e.g., what the business cares about)
  - *System (Designer's View):* Conceptual system design
  - *Technology (Builder's View):* Logical technology design
  - *Detailed (Implementer's View):* Physical design and configuration
  - *Working (User's View):* Actual running system in operation

ZACHMAN FRAMEWORK

- Reification = moving down the rows
  - Reification is the process of making an abstract concept more concrete as you move down the rows
- Example:
  - Consider the “what” or data column
    - *Identification (Scope):* “We need to track customers”
    - *Definition (Business):* “Customer = person with an account”
    - *Representation (System):* “Customer entity in ERD diagram”
    - *Specification (Technology):* “Customer table schema in database”
    - *Configuration (Detailed):* “Configured database table with defined fields”
    - *Instantiation (Working):* “Running production database with real customer records”

ZACHMAN FRAMEWORK

	What (Data)	How (Function)	Where (Locations)	Who (People)	When (Time)	Why (Motivation)
Scope {contextual} Planner	List of things important to the business	List of processes that the business performs	List of locations in which the business operatses	List of organizations important to the business	List of events/ cycles important to the business	List of business goals/strategies
Enterprise Model {conceptual} Business Owner	e.g. Semantic Model	e.g. Business Process Model	e.g. Business Logistics System	e.g. Workflow Model	e.g. Master Schedule	e.g. Business Plan
System Model {logical} Designer	e.g. Logical Data Model	e.g. Application Architecture	e.g. Distributed System Architecture	e.g. Human Interface Architecture	e.g. Process Structure	e.g. Business Rule Model
Technology Model {physical} Implementer	e.g. Physical Data Model	e.g. System Design	e.g. Technology Architecture	e.g. Presentation Architecture	e.g. Control Structure	e.g. Rule Design
Detailed Representation {out-of-context} Subcontractor	e.g. Data Definition	e.g. Program	e.g. Network Architecture	e.g. Security Architecture	e.g. Timing Definition	e.g. Rule Definition
Functioning System	e.g. Data	e.g. Function	e.g. Network	e.g. Organization	e.g. Schedule	e.g. Strategy

ZACHMAN FRAMEWORK

- Application to risk and resilience
- Risks exist at all levels:
  - Abstract
    - For example: unclear business goals introduces the risk of misalignment
  - Concrete
    - For example: poor database backup introduces operational risk
  - Controls map across rows:
    - *Preventive:* high-level standards, policies
    - *Detective:* monitoring at system level
    - *Corrective:* recovery at implementation level
    - The Zachman rows + reification help ensure no blind spots in risk analysis
  - Clarifies how operational systems support higher level functions
    - Assists in developing business impact analysis

OTHER FRAMEWORKS

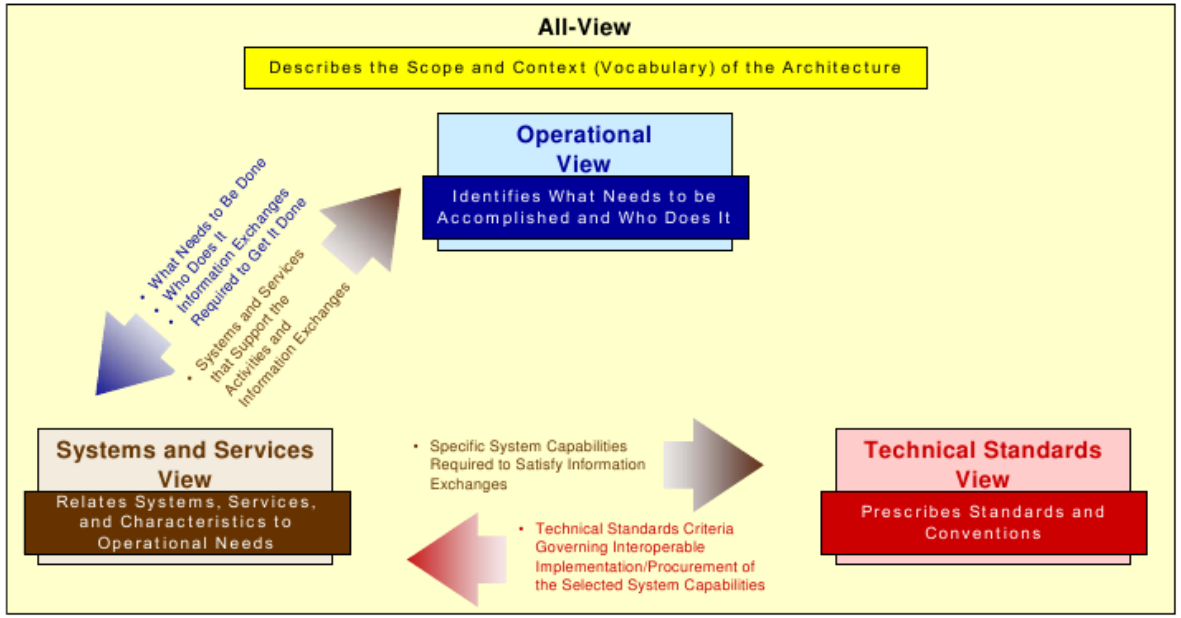
- There are a number of similar frameworks
  - Many of these are used for risk management
  - Again, the primary focus has been cybersecurity
  - TOGAF (The Open Group Architecture Framework)
    - *Origin*: Developed by The Open Group
    - *Focus*: A step-by-step methodology (Architecture Development Method, ADM) for developing and managing enterprise architectures
    - *Strength*: Process-driven, widely adopted in industry, has certifications
    - *Relation to Zachman:* Where Zachman is a taxonomy (classification scheme), TOGAF provides a process to build architecture. They are often used together

OTHER FRAMEWORKS

- FEAF (Federal Enterprise Architecture Framework)
  - *Origin*: Developed by the U.S. Federal CIO Council
  - *Focus*: Standardized approach for U.S. government agencies to align IT with business goals
  - *Strength*: Provides reference models (Performance, Business, Service, Data, Technical)
  - *Relation to Zachman*: Like Zachman, it provides structured “views” of architecture, but is tailored to compliance and governance in public sector

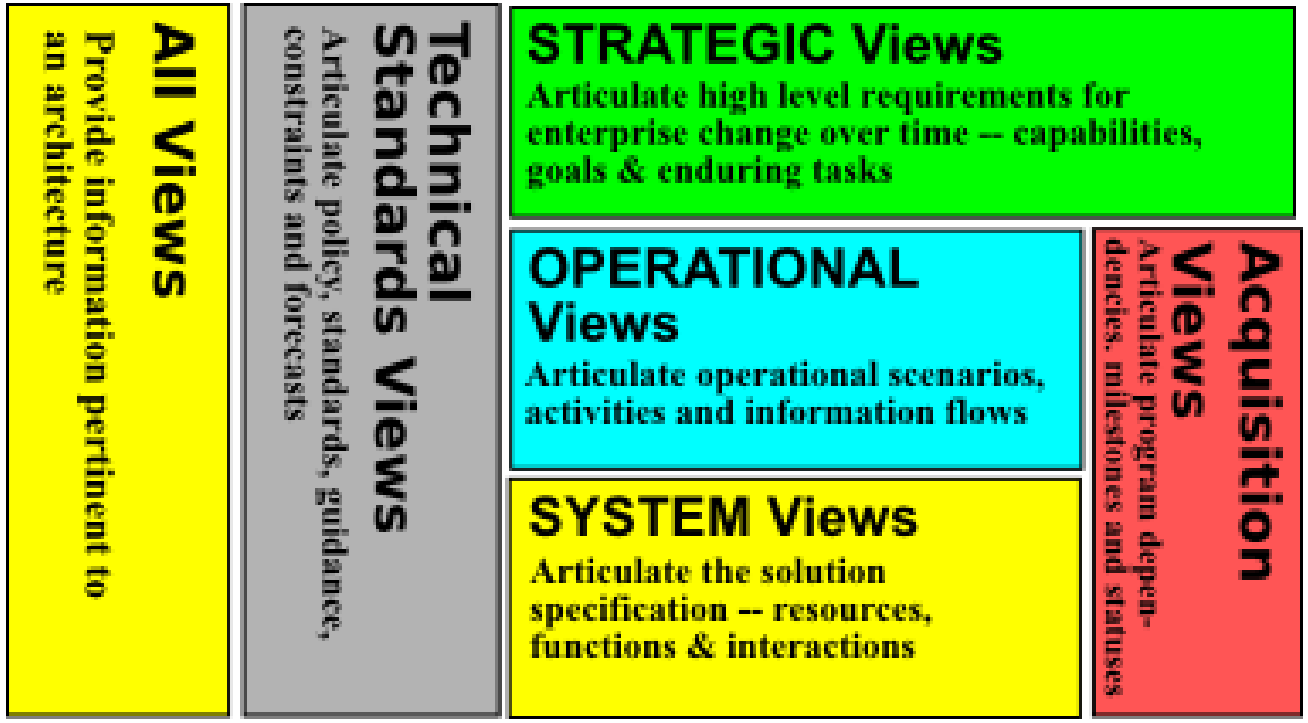
OTHER FRAMEWORKS

- DoDAF (Department of Defense Architecture Framework)
  - *Origin*: U.S. Department of Defense.
  - *Focus*: Architectural views for defense systems; emphasizes operational, system, and technical standards
  - *Strength*: Highly structured, supports complex mission systems, risk management, and interoperability
  - *Relation to Zachman*: Similar multi-view approach but specialized for defense/resiliency contexts



OTHER FRAMEWORKS

- MODAF (Ministry of Defence Architecture Framework)
  - *Origin*: UK Ministry of Defence (similar to DoDAF)
  - *Focus*: Defense enterprise architecture, interoperability, and mission resiliency
  - *Strength*: Structured views and models, later influenced international standards
  - *Relation to Zachman*: Zachman's conceptual rigor influenced its grid-style structuring



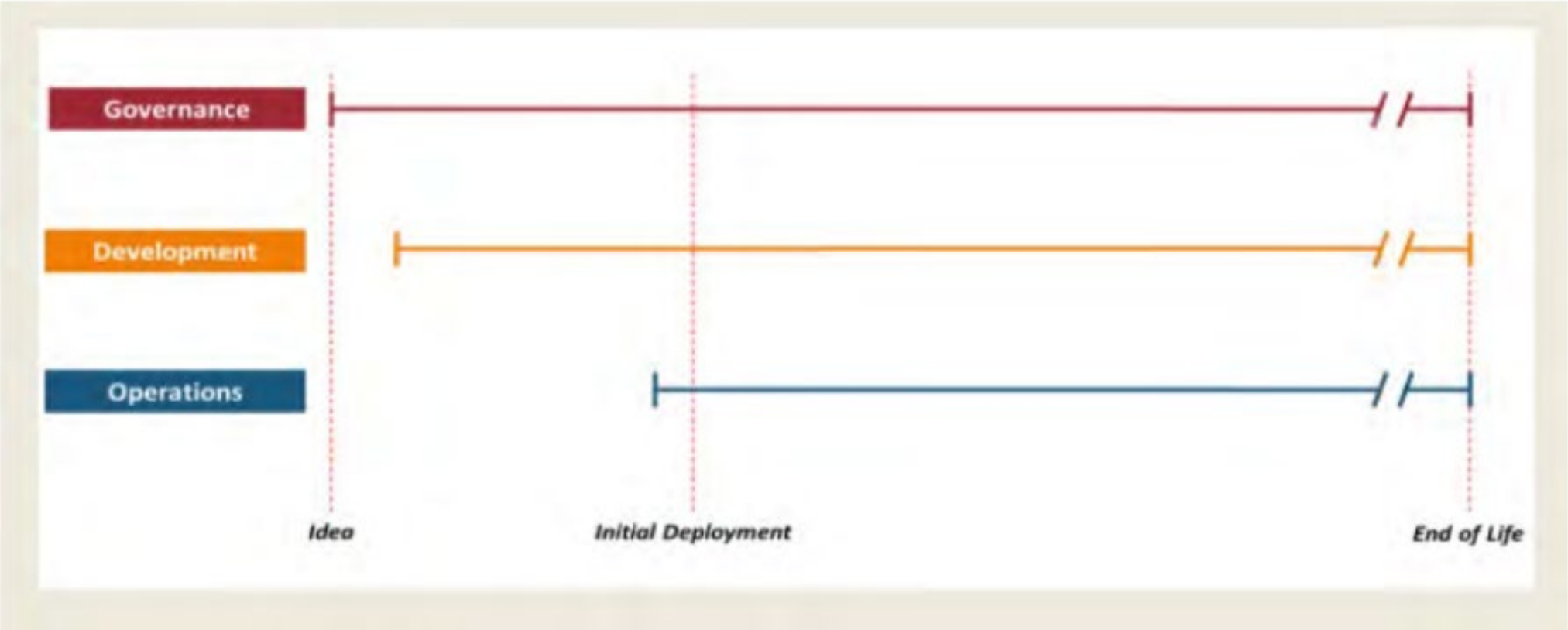
OTHER FRAMEWORKS

- ISO/IEC/IEEE 42010 (Standard for Architecture Description)
  - *Origin*: Joint ISO/IEC/IEEE standard
  - *Focus*: Provides a standard way to describe architectures with viewpoints, stakeholders, concerns
  - *Strength*: Internationally recognized; focuses on architecture descriptions rather than processes
  - *Relation to Zachman*: Shares the idea of viewpoints and stakeholder perspectives as in Zachman's rows and columns

APPLICATION LIFECYCLE MANAGEMENT

- ALM analyzes software as a product with a lifecycle, not just code to be delivered once
  - It integrates three continuous streams:
    - *Governance*: oversight, compliance, risk management, business alignment
    - *Development*: building, enhancing, testing, and integrating the software
    - *Operations*: running, monitoring, supporting, and maintaining the application in production
  - These streams span across the major milestones of an application's lifetime:
    - *Initiation*: concept, business case, requirements, governance setup
    - *Deployment*: release into production and ongoing updates
    - *Retirement*: phasing out, migrating data, decommissioning

APPLICATION LIFECYCLE MANAGEMENT



THE THREE STREAMS OF ALM

- Governance
  - Ensures the application aligns with business goals, compliance, and risk appetite
  - Defines policies for security, resiliency, quality standards, and regulatory adherence
  - Weak governance means resilience goals (e.g., recovery time, data protection) may never get built into requirements
- Development
  - Covers design, coding, testing, integration, and continuous delivery
  - Evolves applications over time, introducing features and bug fixes
  - Poor development discipline creates technical debt, security flaws, and fragile code and increasing resilience risk

THE THREE STREAMS OF ALM

- Operations
  - Runs the software in real-world environments with monitoring, support, and maintenance
  - Focused on uptime, stability, patching, and performance
  - Operational blind spots (e.g., weak monitoring, poor incident response) lead to prolonged outages
- How ALM supports risk and resilience
  - *Governance*: Ensures that resilience goals (uptime, compliance, recovery models) are part of strategy from day one
  - *Development*: Embeds resilience into code (error handling, modular design, secure development)
  - *Operations*: Delivers resilience in practice (monitoring, failover, recovery, incident response)
  - *Milestones*: Provide checkpoints where risk can be assessed and controls adjusted

LIFECYCLE MILESTONES

- Initiation: start of life
  - Business case approved, governance structure established, resilience requirements set (e.g., uptime, recovery objectives)
  - Risk if skipped:
    - Misalignment with business continuity; resilience needs bolted on later instead of designed in
- Deployment: operational life
  - Application released and evolves under DevOps / CI/CD processes
  - Operations ensure stability while governance checks compliance and risk exposure
  - Risk if weak:
    - Frequent outages, compliance failures, missed SLAs

LIFECYCLE MILESTONES

- Retirement: end of life
  - Application decommissioned with data migrated, dependencies untangled, and security risks removed
  - Risk if ignored:
    - Legacy risk persists: “zombie apps” still running unsupported, exposing the enterprise to resilience failures
- Zachman and ALM
  - Zachman shows views of architecture
  - ALM shows the lifecycle of applications
  - Together they help identify risk and resilience issues across time and across system dimensions

Q&A AND OPEN DISCUSSION