Accelerating Software Transformation

The Synergy between Domain-Driven Design & Systems Thinking

Masoud Chelongar
Hands-on Software Architect

Overview

Defining The Problem Space

Overview

Defining The Problem Space

Investigating Tools & Methodologies

Overview

Defining The Problem Space

Investigating Tools & Methodologies

Designing The Solution Framework

What is Software Transformation

✓ Evolving an Organisation's Software

What is Software Transformation

- ✓ Evolving an Organisation's Software
- ✓ Adopt Software to The New Business Requirements

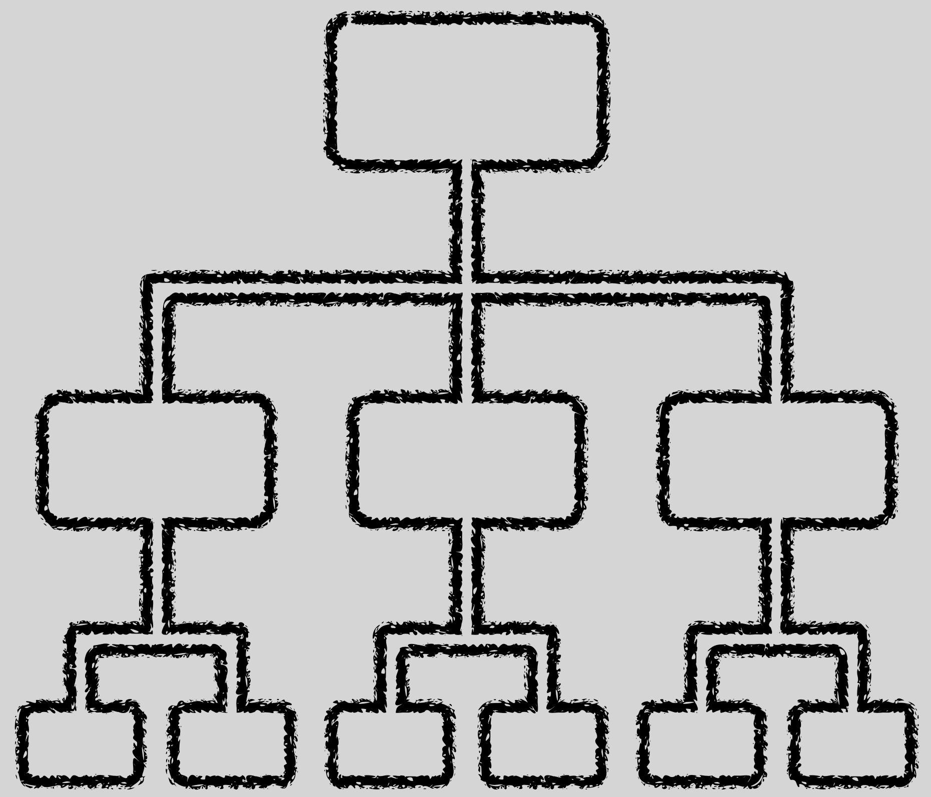
What is Software Transformation

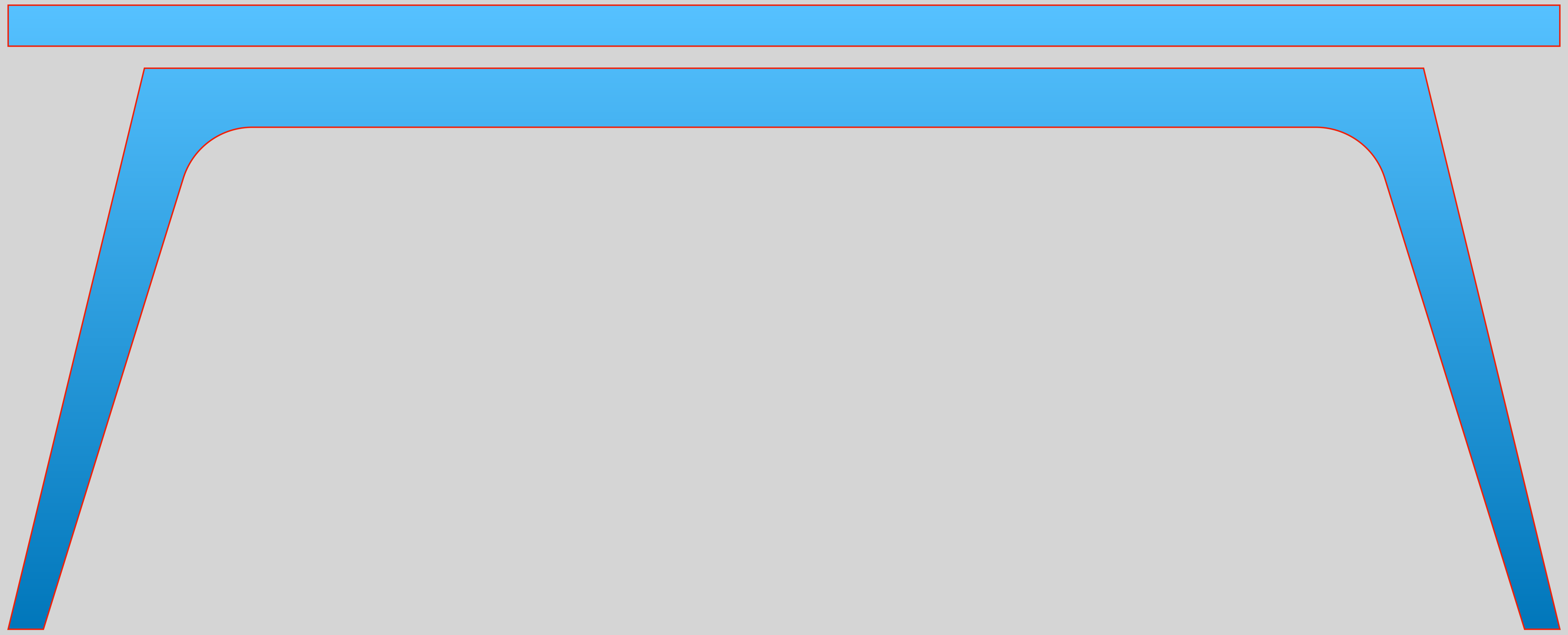
- ✓ Evolving an Organisation's Software
- ✓ Adopt Software to The New Business Requirements
- ✓ Change The Software Based on Market Demands

Failing to act
evolutionary
compels revolutionary action.

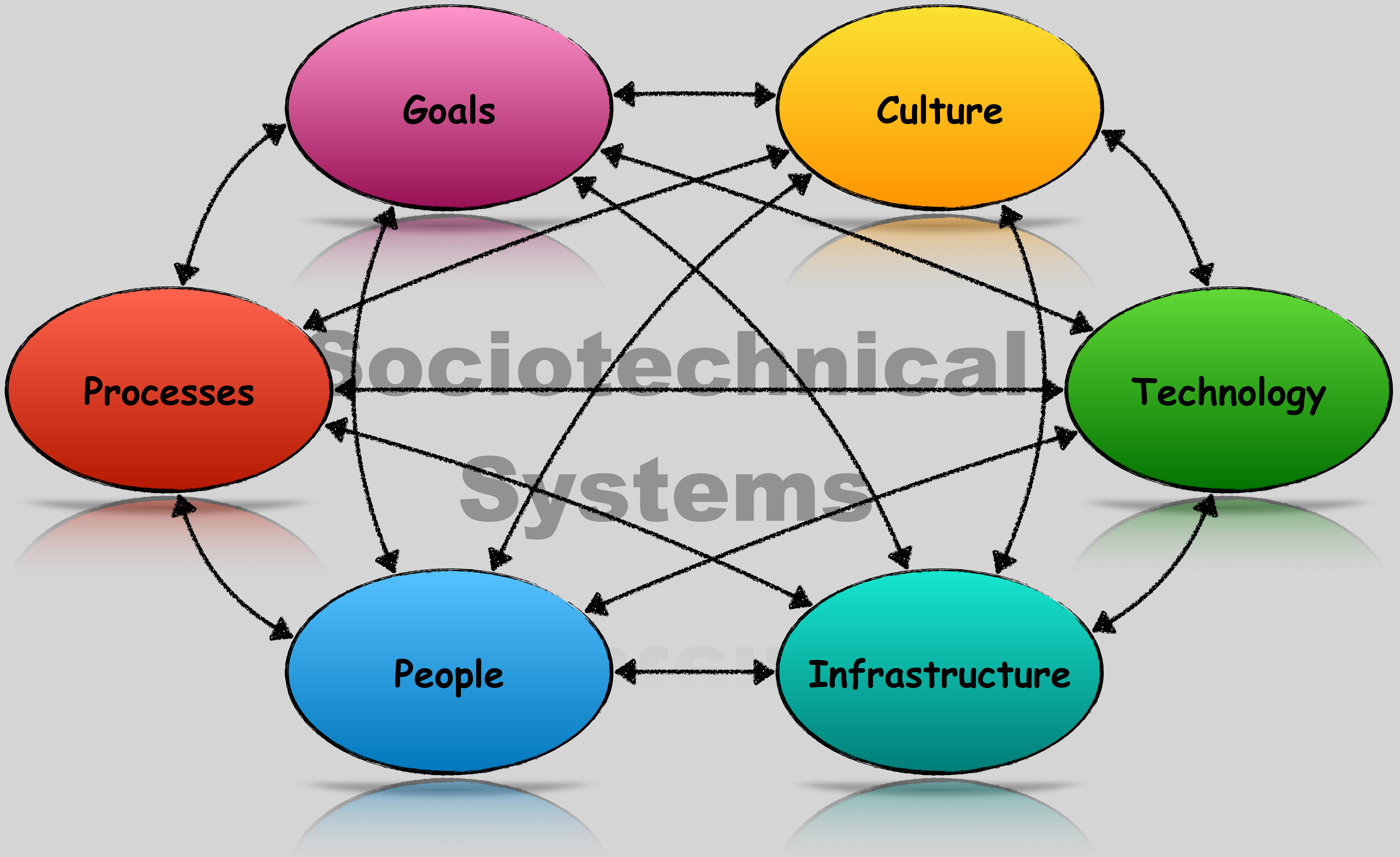
Technical & Organisational

&





Sociotechnical Systems


Goals
Culture
Processes
Sociotechnical Systems
Technology
People
Infrastructure

People

Skills Gap

Leadership Buy-in

Resistance To Change

Collaboration Challenges

Infrastructure

Legacy Systems

Integration Issues

Scalability Concerns

Downtime Risk

Culture

- Cultural Misfits
- Fear of Accountability
- Motivation Challenges
- Cross-Functional Tension

Processes

- Misalignment of Processes & Tools
- Documentation Deficiency
- Change Management

Technology

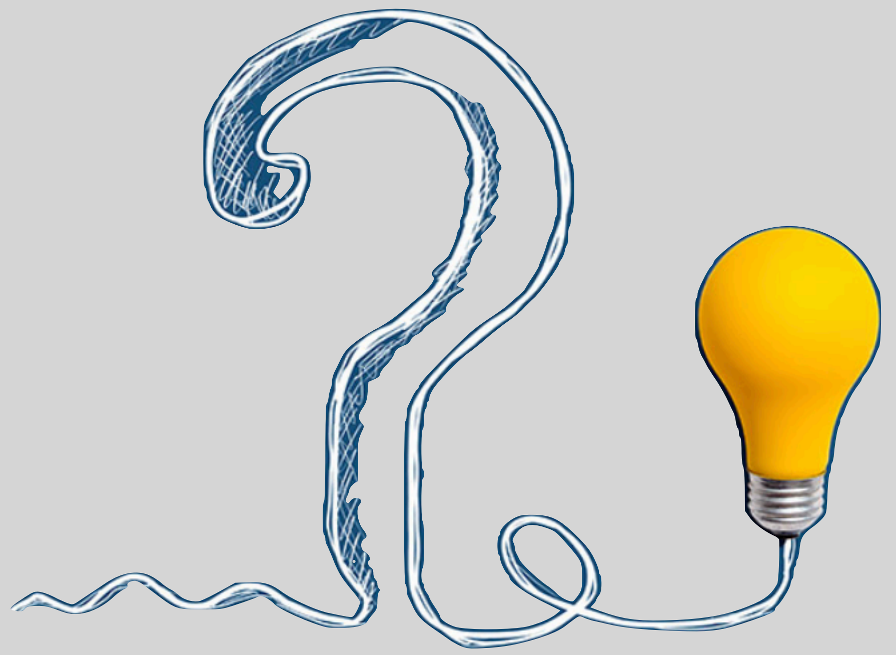
- Customisation & Standardisation
- Rapidly Changing Landscape
- Security Concerns
- Vendor Lock-in

Goals

- Ambiguity in Objectives
- Short-Term Focus
- KPIs Mismatch
- Stakeholder Alignment

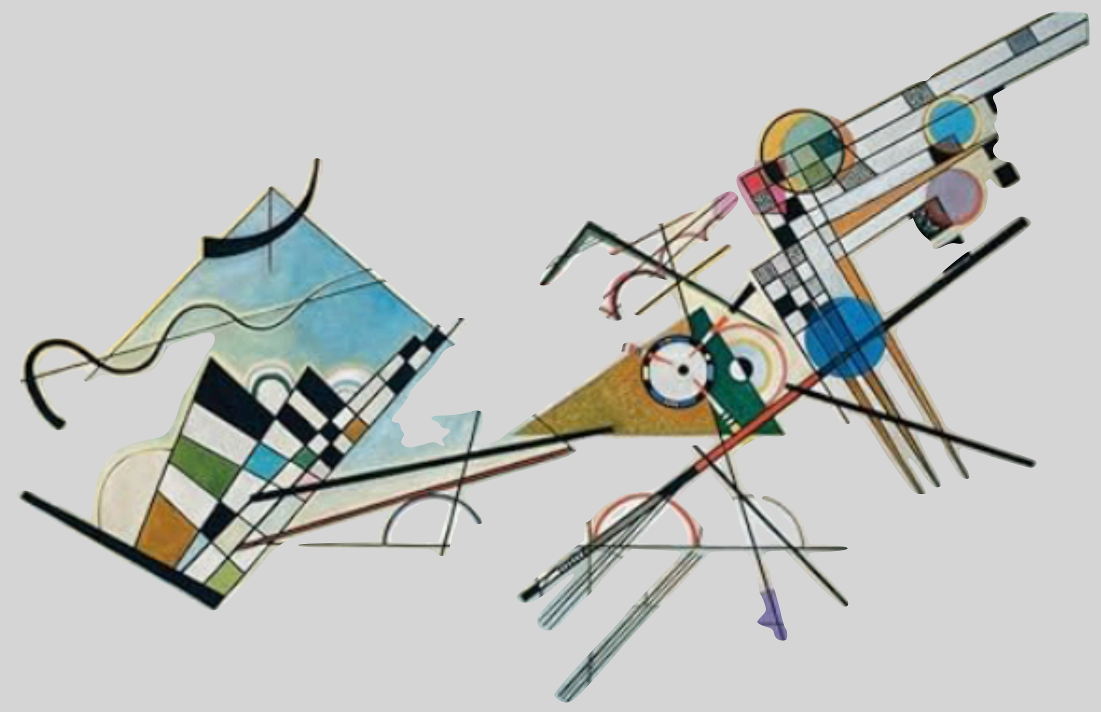
IDEALISM
REALISM

Hybrid Solution



+

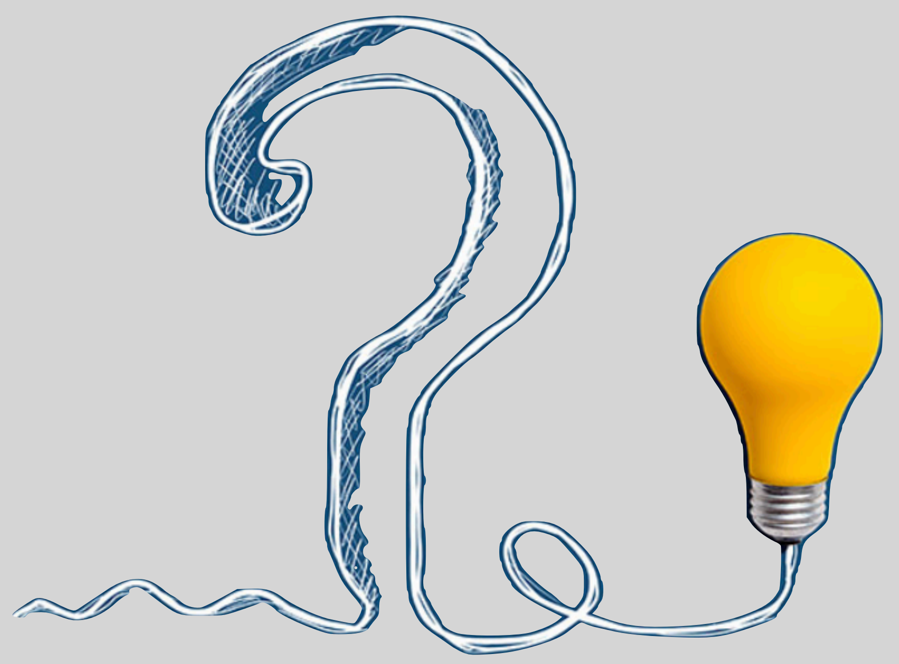
+



Cross-Disciplinary Thinking

Team Dynamics

Business-Centric Development

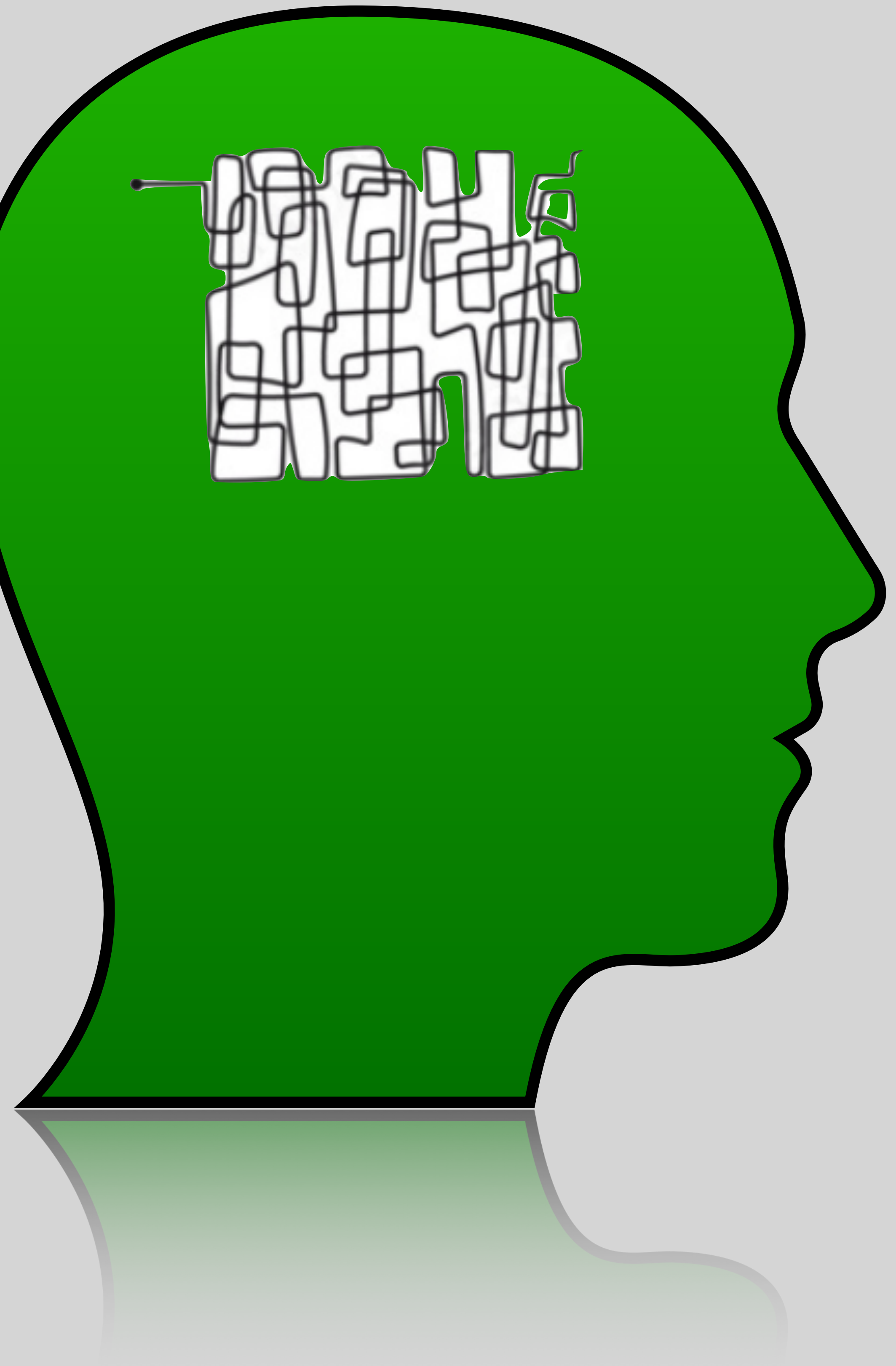


Cross-Disciplinary Thinking

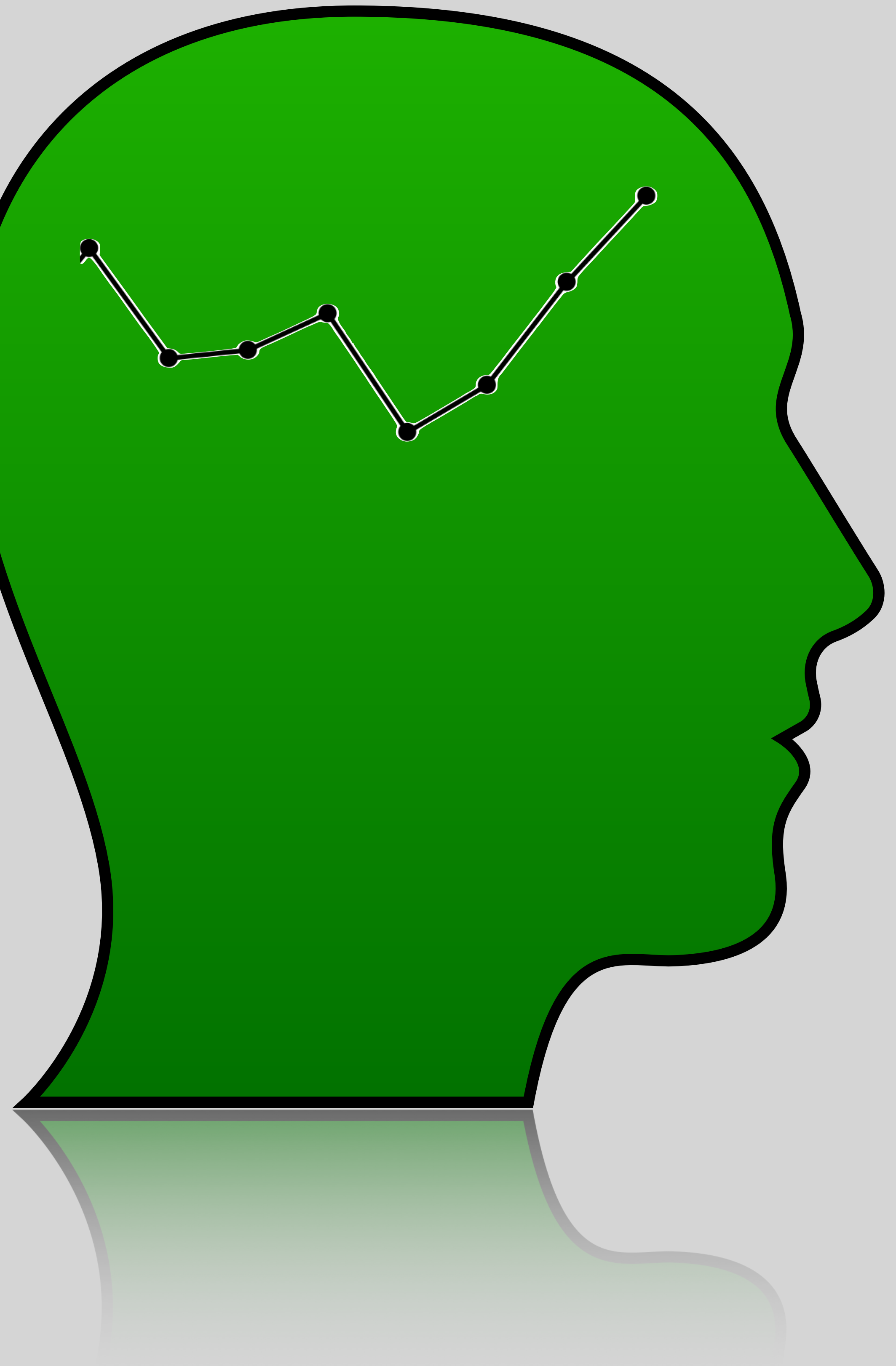
In the Systems Age we tend to look at things as part of larger wholes rather than as wholes to be taken apart.

—Russell L. Ackoff

Think Non-Linearly

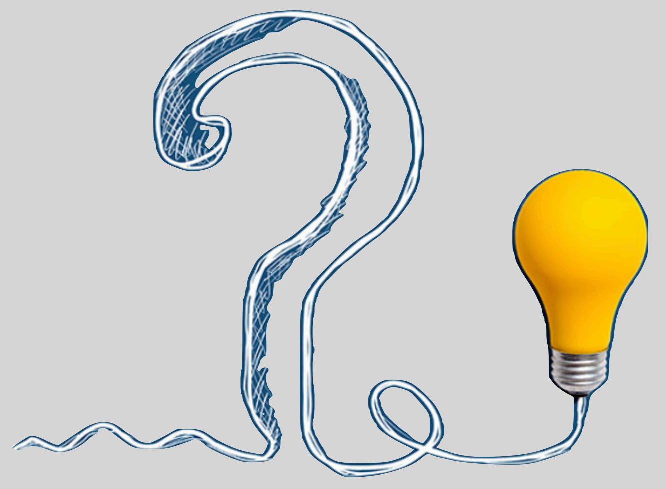


vs.

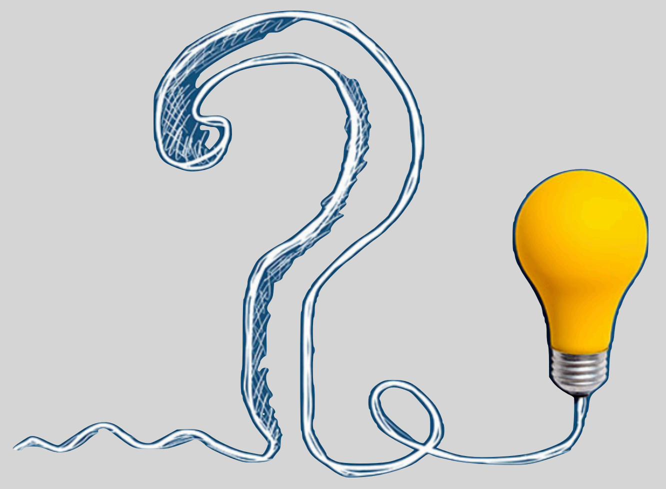


Linear Thinking

Procedural



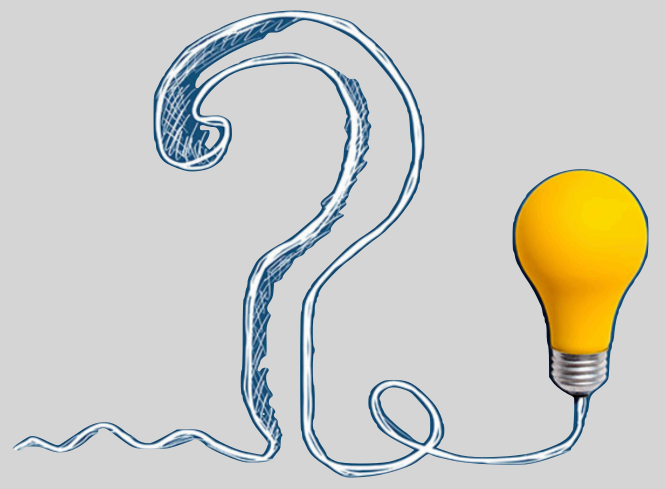
Linear Thinking



Procedural

Predictable Process

Linear Thinking

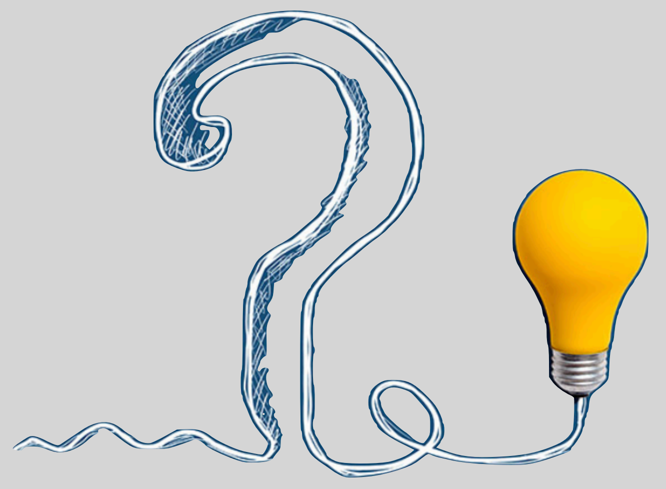


Procedural

Predictable Process

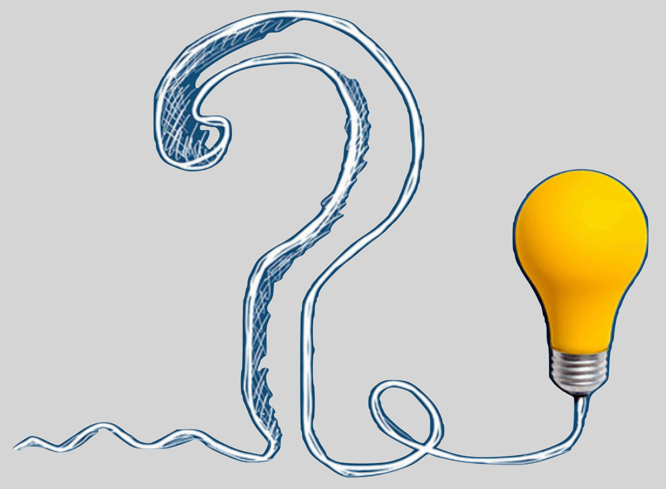
Step-by-Step Approach

Linear Thinking



- Procedural
- Predictable Process
- Step-by-Step Approach
- Focused Problem Solving Approach

Linear Thinking



Procedural

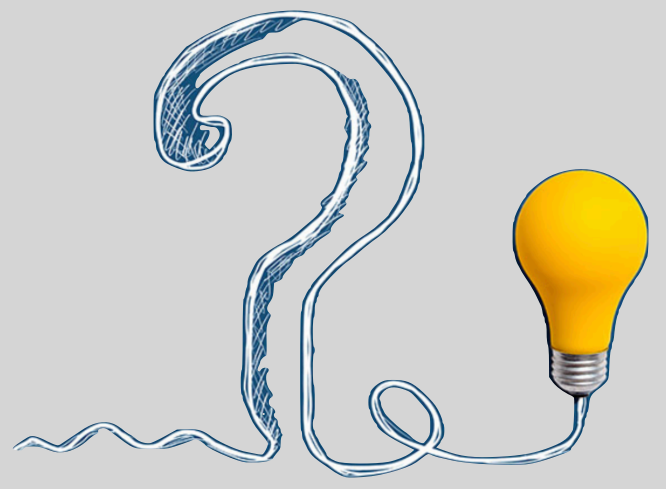
Predictable Process

Step-by-Step Approach

Focused Problem Solving Approach

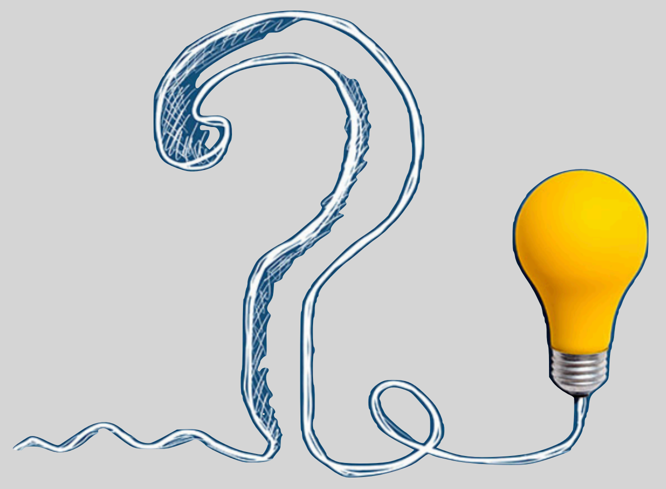
Essential Well-Defined Requirements

None-Linear Thinking



Iterative Process

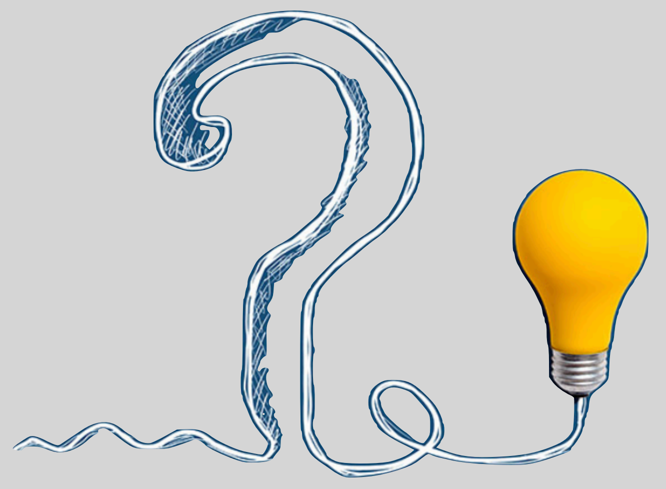
None-Linear Thinking



Iterative Process

User-centric Development

None-Linear Thinking

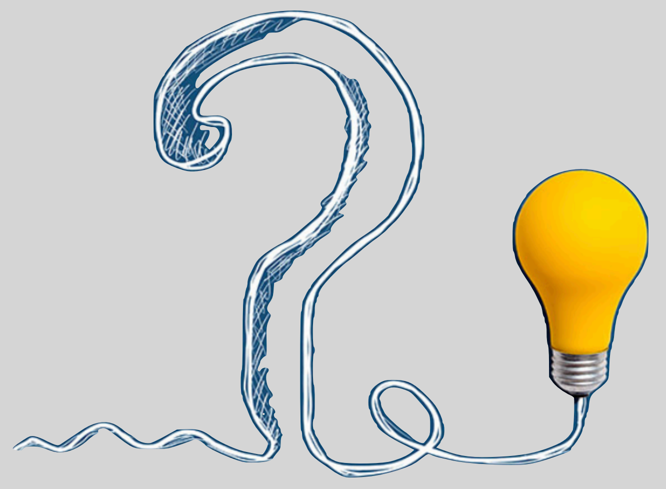


Iterative Process

User-centric Development

Parallel Problem Solving

None-Linear Thinking



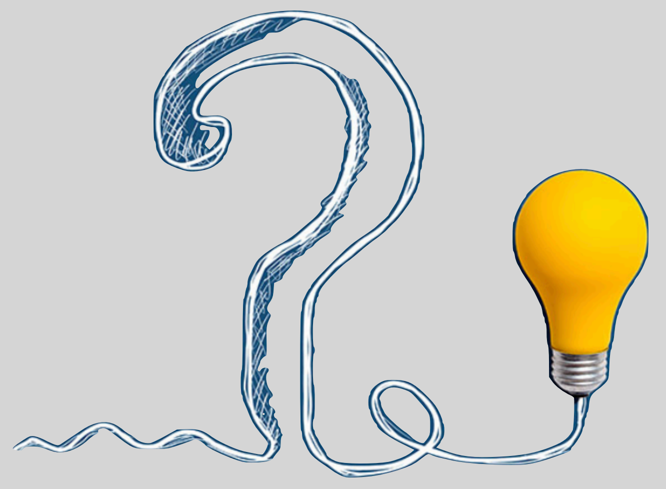
Iterative Process

User-centric Development

Parallel Problem Solving

Based on Incremental Development

None-Linear Thinking



Iterative Process

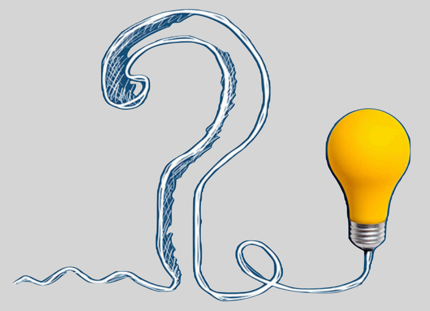
User-centric Development

Parallel Problem Solving

Based on Incremental Development

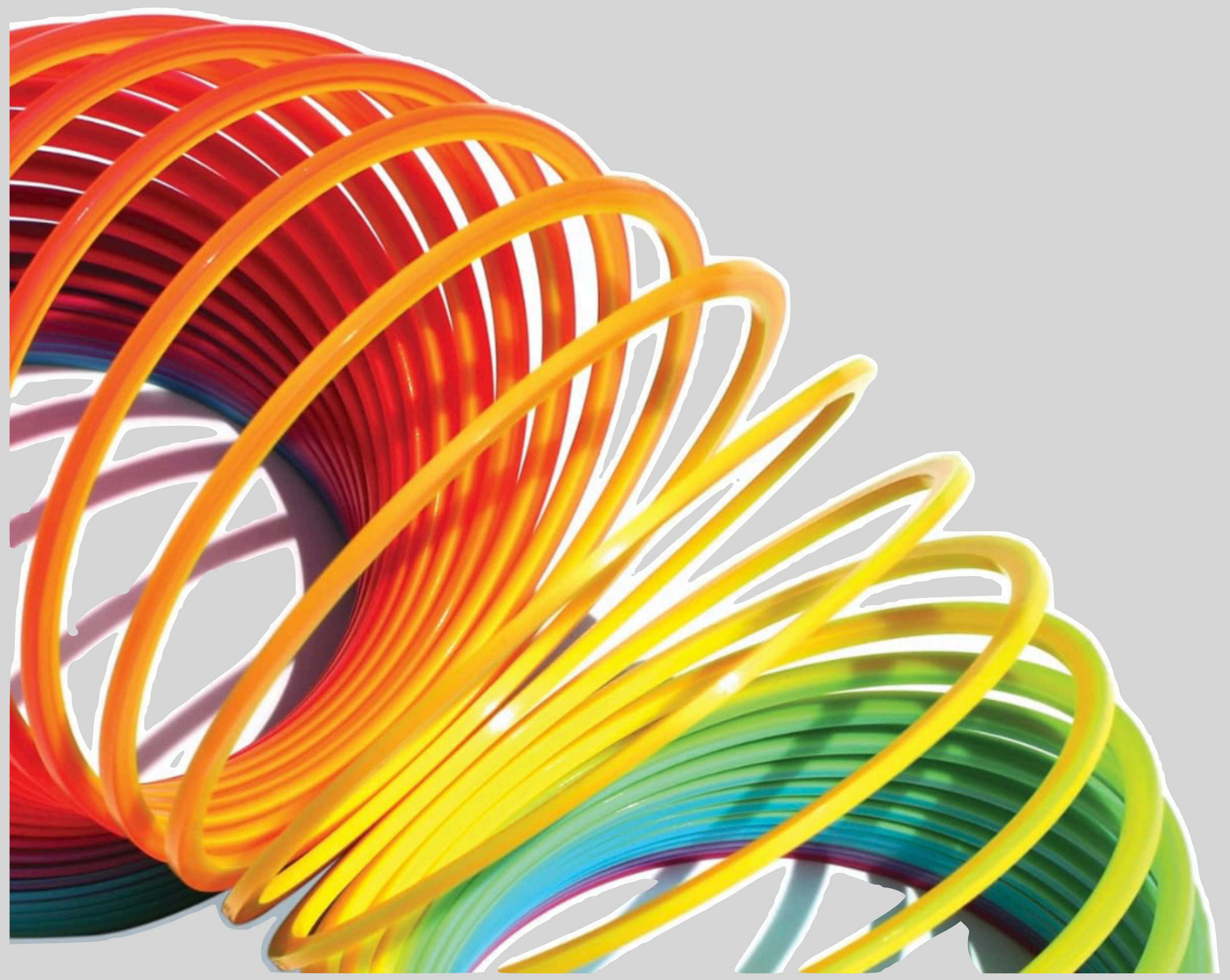
Focus on Collaboration and Communication

Systems Thinking

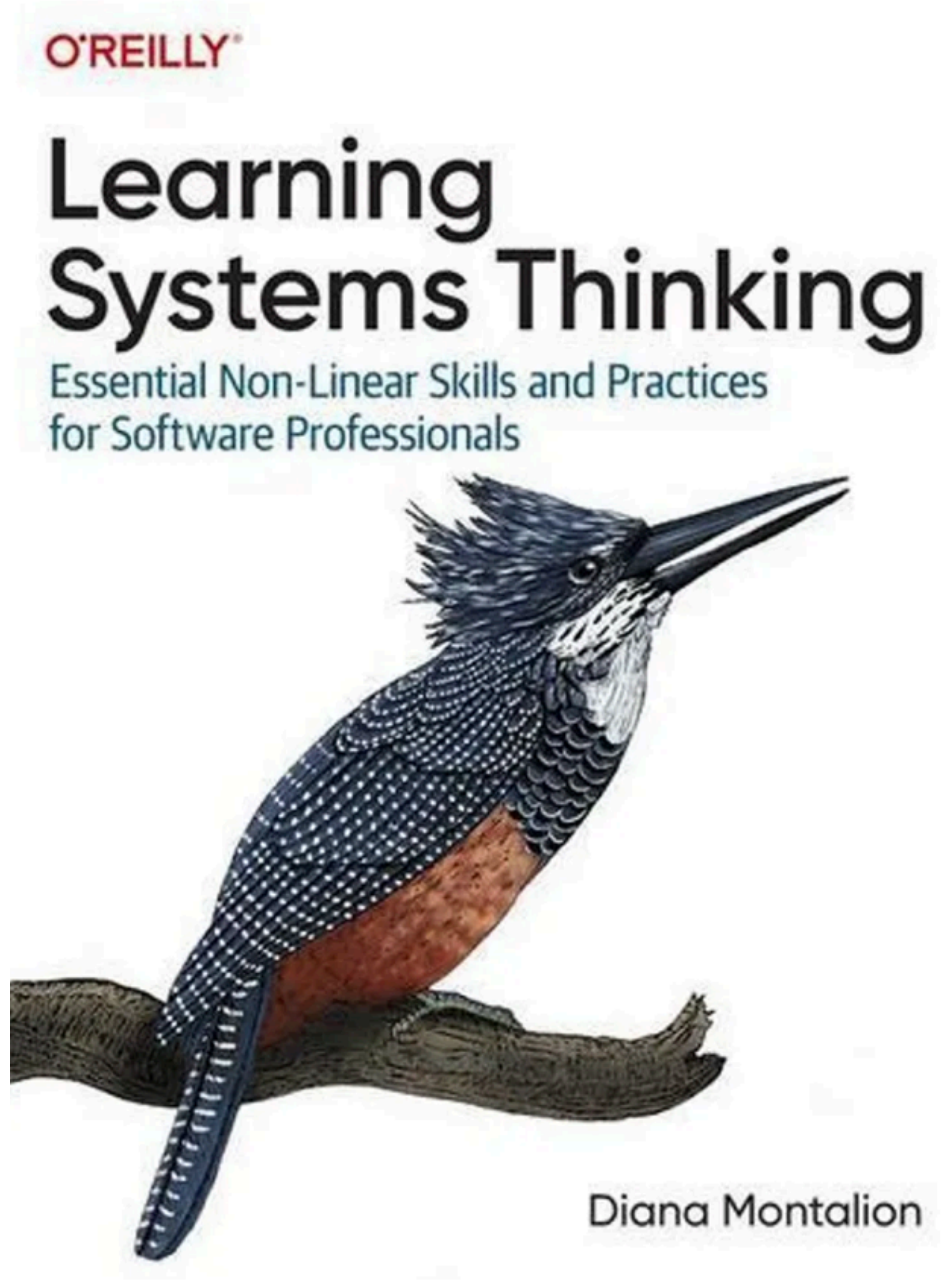


For those who stake their identity on the role of omniscient conqueror, the uncertainty exposed by systems thinking is hard to take. If you can't understand, predict, and control, what is there to do?

—Donella Meadows, Thinking in System



Systems Thinking


O'REILLY
Learning Systems Thinking
Essential Non-Linear Skills and Practices for Software Professionals
Diana Montalion

Systems thinking expands our toolsets as knowledge workers. It steps us outside the constant, pointless culture war about architecture versus engineering as a practice

—Diana Montalion, Learning Systems Thinking

Team Dynamics

Team Dynamics

Role Clarification

Team Dynamics

Role Clarification

Foster Collaboration

Team Dynamics

Role Clarification

Foster Collaboration

Reduced Process Frictions

Team Dynamics

Role Clarification

Foster Collaboration

Reduced Process Frictions

Promoting Ownership & Aligning Values

Team Dynamics

Role Clarification

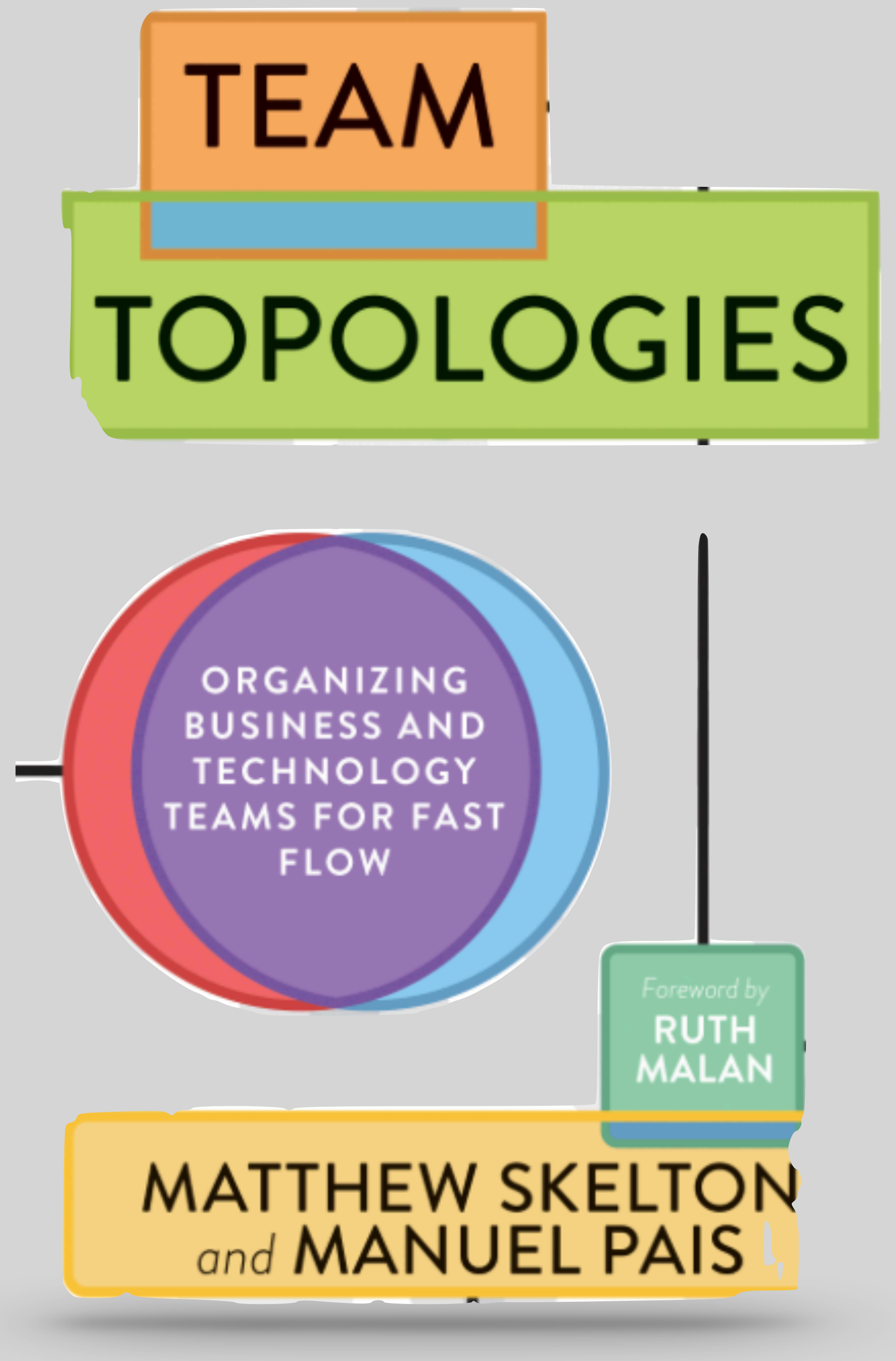
Foster Collaboration

Reduced Process Frictions

Promoting Ownership & Aligning Values

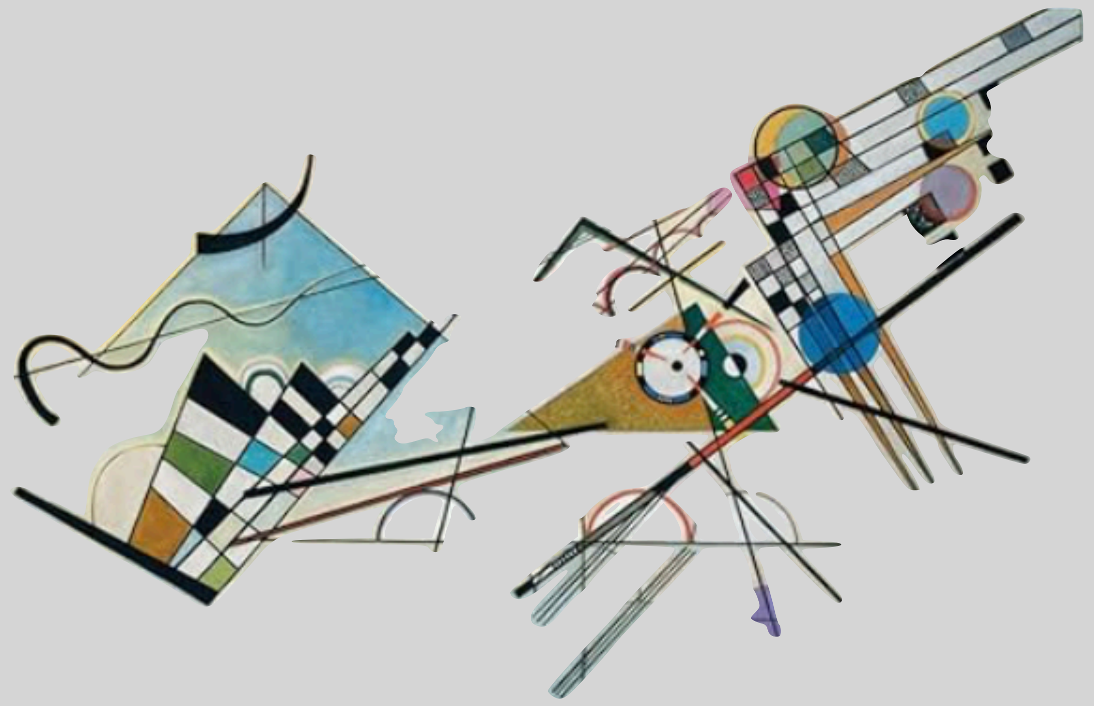
Encouraging Continues Learning Culture

Team Topologies


TEAM
TOPOLOGIES
ORGANIZING BUSINESS AND TECHNOLOGY TEAMS FOR FAST FLOW
Foreword by RUTH MALAN
MATTHEW SKELTON and MANUEL PAIS

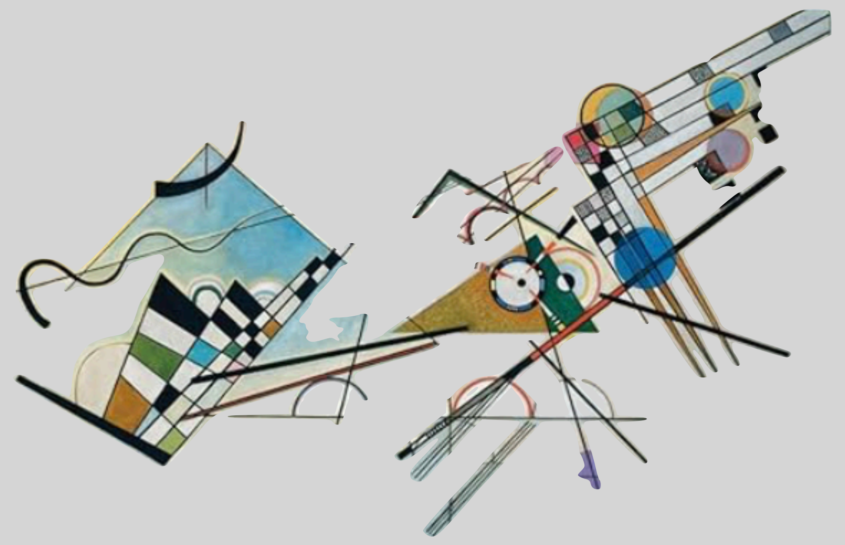
Team Topologies is an approach to designing team-of-teams organisations for fast flow of value.

—Manuel Pais & Matthew Skelton



Business-Centric Development

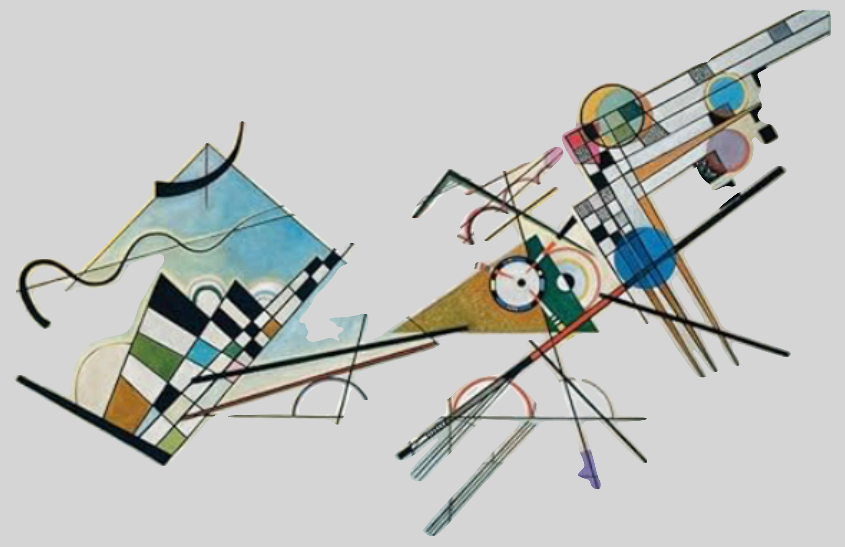
Business-Centric Development



Technology

- ❖ Choose technology stack that reflects business needs

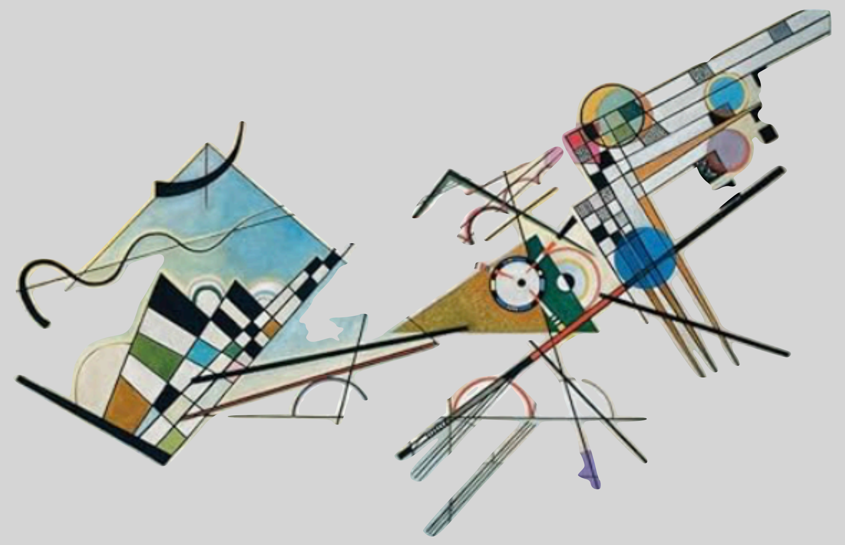
Business-Centric Development



Processes

- ❖ Define aligned workflows
- ❖ Establish efficient communication or clear cross-team dependencies

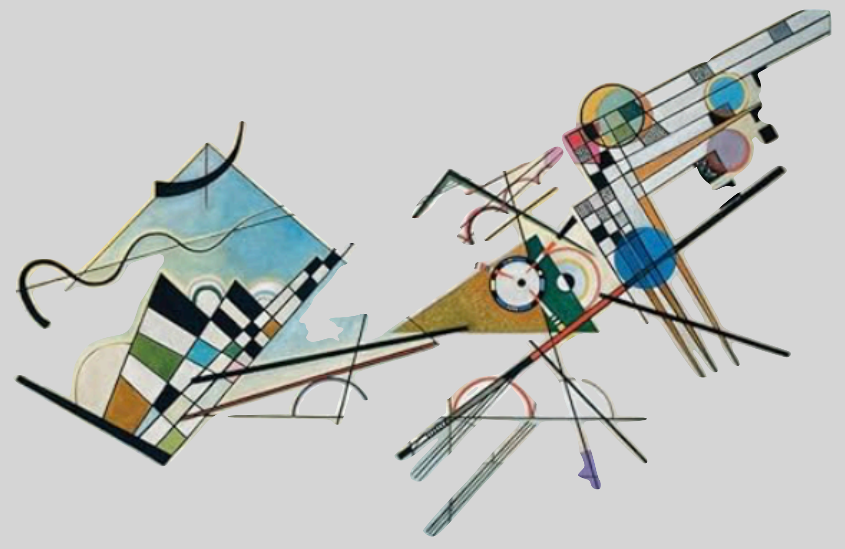
Business-Centric Development



People

- ❖ Align suitable teams
- ❖ Define clear ownership

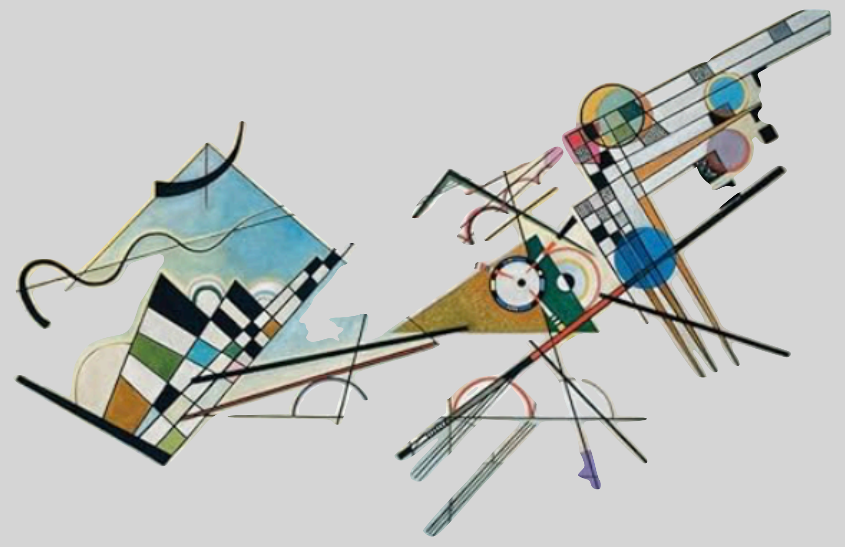
Business-Centric Development



Culture

- ❖ Cultural silos Prevention
- ❖ Cultivate strong communication between technical and business teams

Business-Centric Development



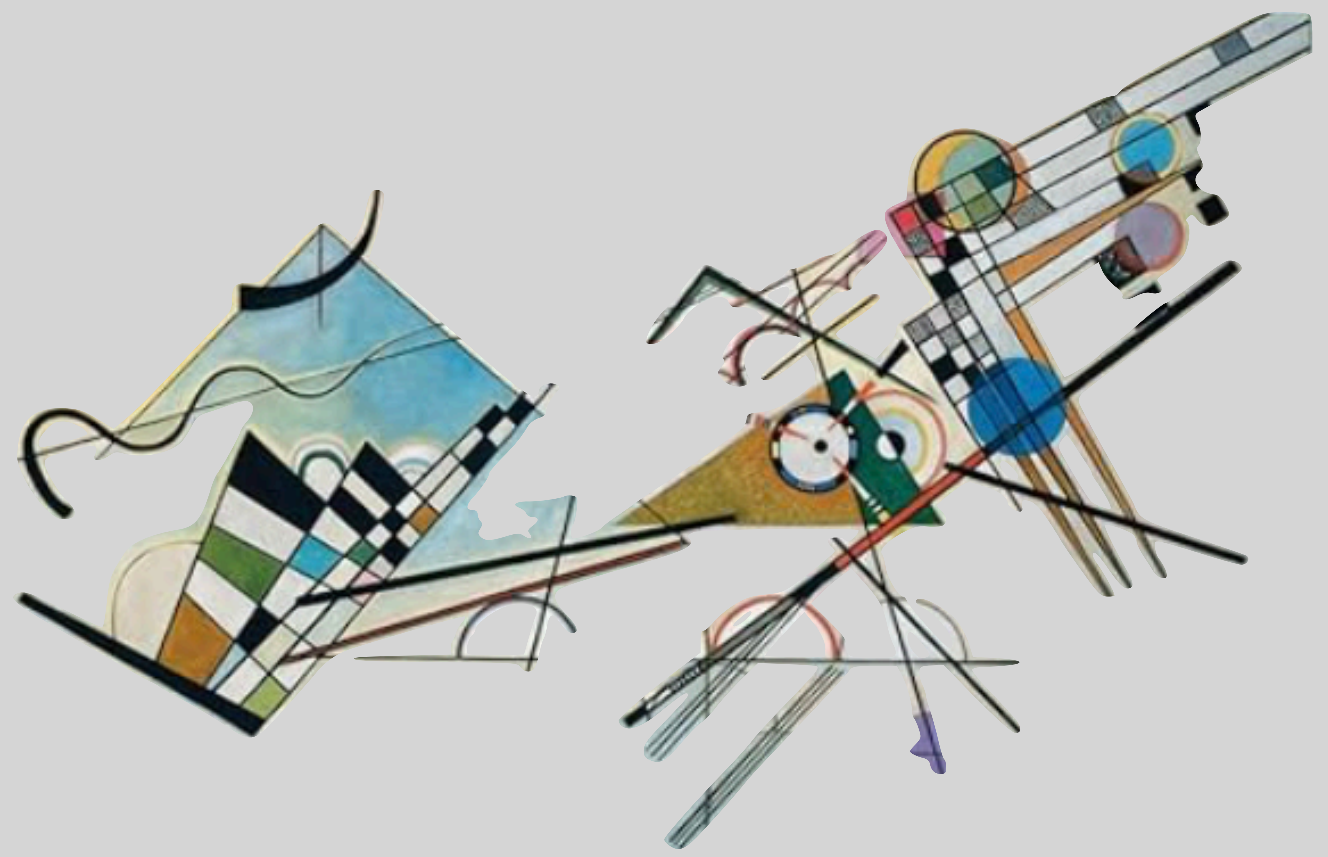
Goal

- ❖ Align objectives
- ❖ Scattered priorities or poorly defined success metrics prevention

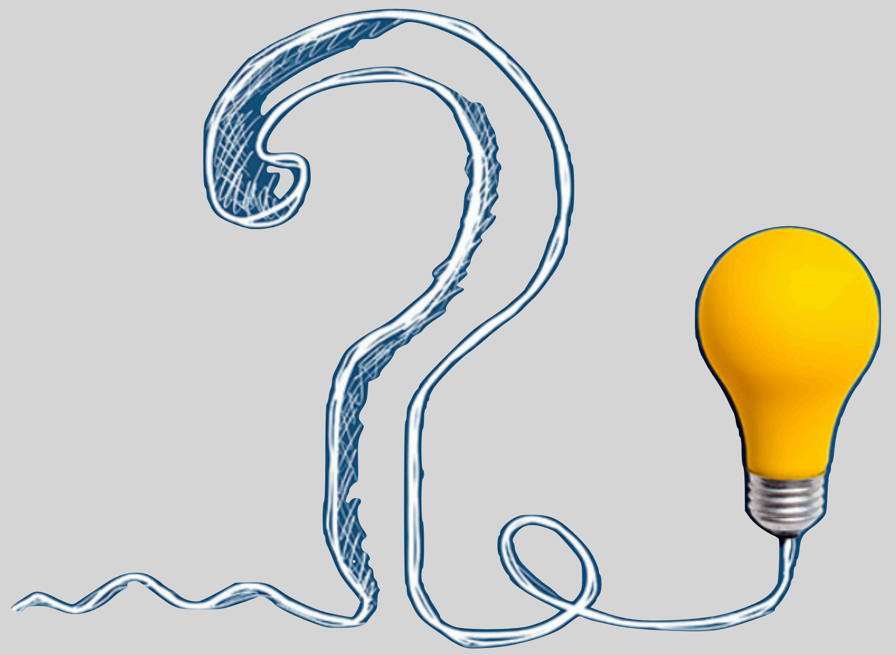
Domain-Driven Design

Domain-Driven Design is an approach to software development that centres the development on programming a domain model that has a rich understanding of the processes and rules of a domain.

—Martin Fowler

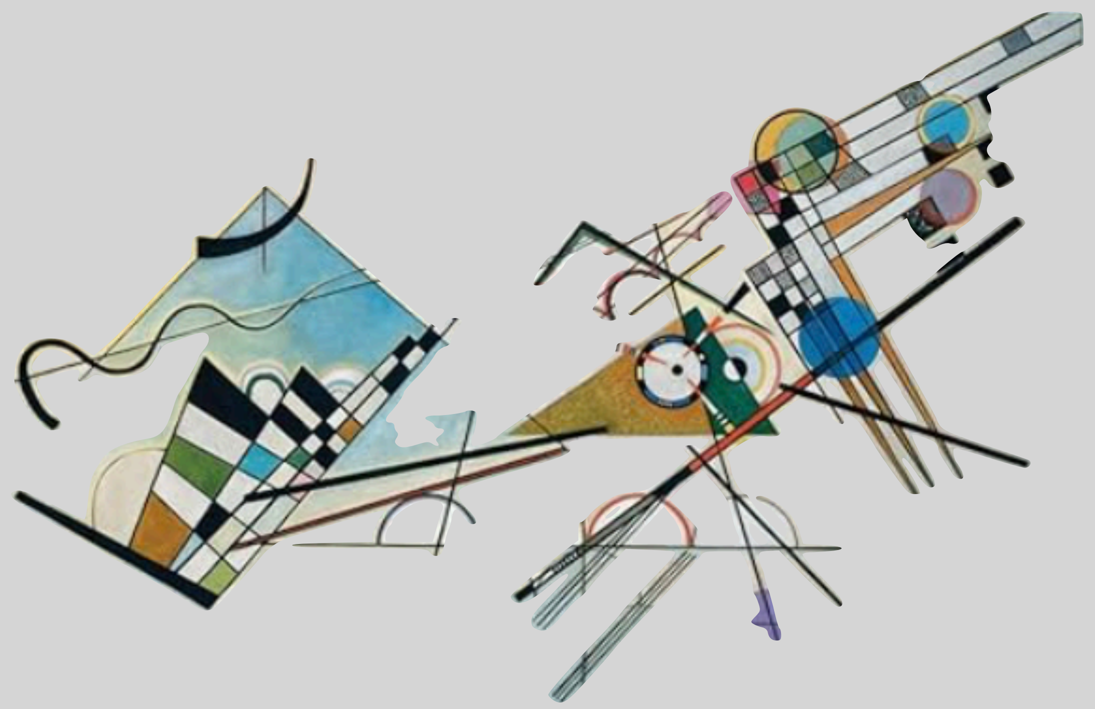


Hybrid Solution



+

+



Systems Thinking

Team Topologies

Domain-Driven Design

Step 1

Define The Whole Picture With Systems Thinking

Analyse Feedback Loops

- Identify reinforcing (Positive) and balancing (Negative) loops

1

Identify System Boundaries

- Map out the entire Sociotechnical System
- Apply Casual loop diagram and stock-and-flow model to understand dependencies

Analyse Feedback Loops

- Identify reinforcing (Positive) and balancing (Negative) loops

1

Analyse Feedback Loops

- ✤ Identify reinforcing (Positive) and balancing (Negative) loops

Identify System Boundaries

- ✤ Map out the entire Sociotechnical System
- ✤ Apply Casual loop diagram and stock-and-flow model to understand dependencies

Identify & Focus on Leverage Points

- ✤ Pinpoint high-impact areas where small changes can drive significant improvements.

Analyse Feedback Loops

✤ Identify reinforcing (Positive) and balancing (Negative) loops

Identify System Boundaries

✤ Map out the entire Sociotechnical System

✤ Apply Casual loop diagram and stock-and-flow model to understand dependencies

Set SW Transformation Goals

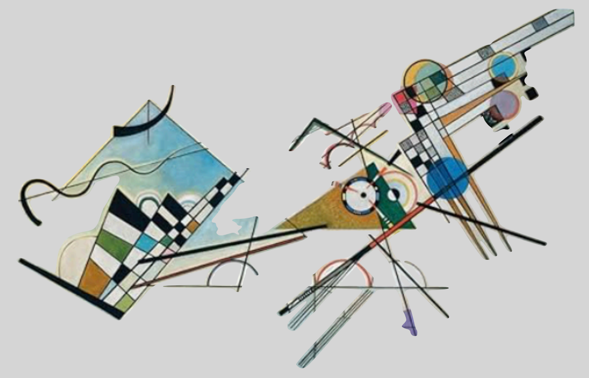
✤Align goals across all six sociotechnical system groups
✤ Use end-to-end cycle time or customer satisfaction metrics to reflect system-wide outcomes

Identify & Focus on Leverage Points

✤ Pinpoint high-impact areas where small changes can drive significant improvements.

Step 2

Structure The Problem Space With Domain-Driven Design



Identify Domains and Bounded Contexts

- Core, Supporting, and Generic Domains
- Define Bounded Contexts for each domain

Model Domains

- ✤ Use Entities, Aggregates, Value Objects, and Event Storming to model domain logic

2

Identify Domains and Bounded Contexts

- ✤ Core, Supporting, and Generic Domains
- ✤ Define Bounded Contexts for each domain

Model Domains

- ✤ Use Entities, Aggregates, Value Objects, and Event Storming to model domain logic

Identify Domains and Bounded Contexts

- ✤ Core, Supporting, and Generic Domains
- ✤ Define Bounded Contexts for each domain

Design Context Maps

- ✤ Map relationships and interactions between Bounded Contexts
- ✤ Define map to plan how domains will interact during and after the transformation

Model Domains

- ✤ Use Entities, Aggregates, Value Objects, and Event Storming to model domain logic

2

Identify Domains and Bounded Contexts

- ✤ Core, Supporting, and Generic Domains
- ✤ Define Bounded Contexts for each domain

Design Context Maps

- ✤ Map relationships and interactions between Bounded Contexts
- ✤ Define map to plan how domains will interact during and after the transformation

Focus on Core Domains

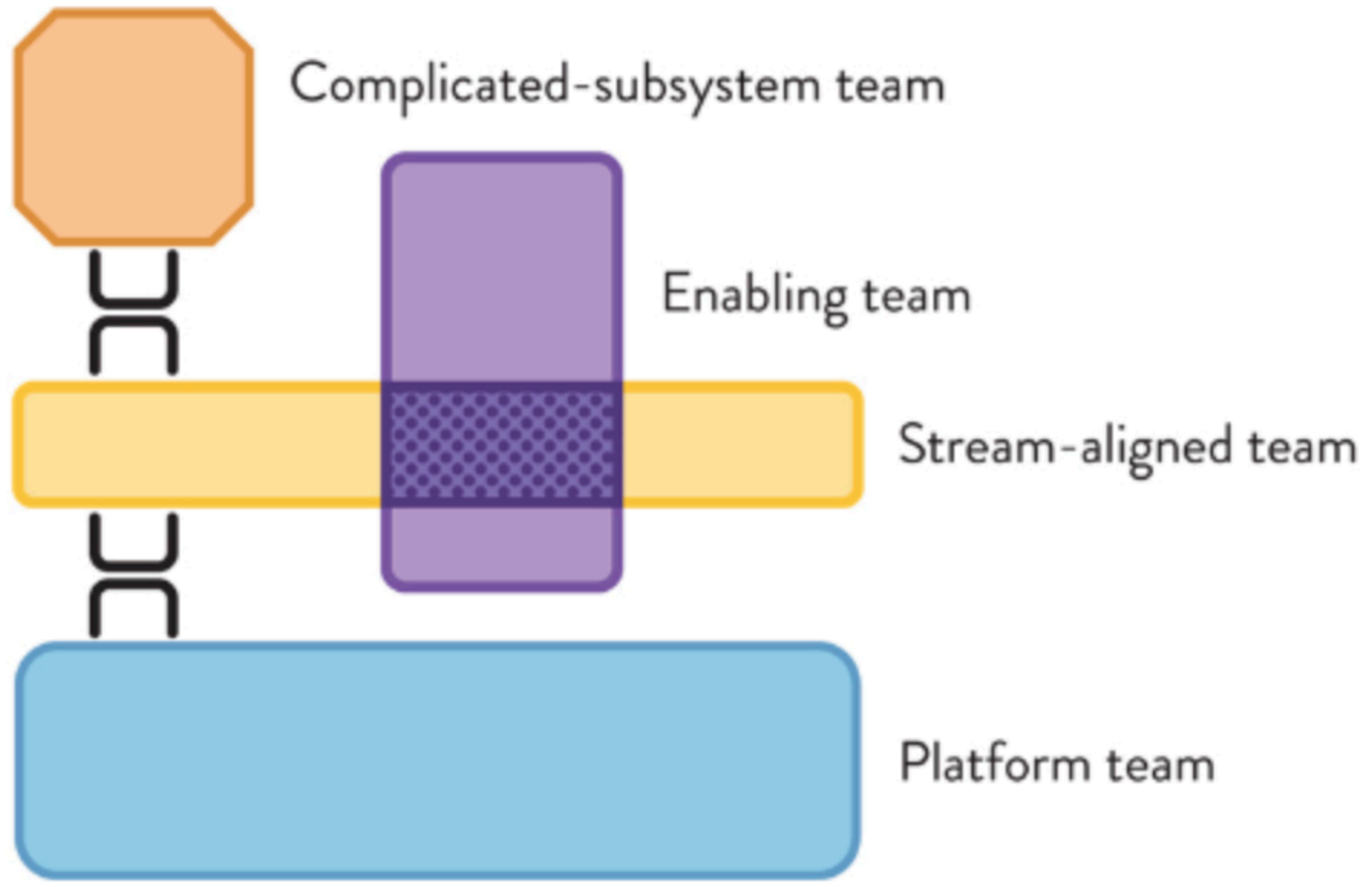
- ✤ Prioritise transformation efforts around the core domains

Step 3

Optimise Organisation Design With Team Topologies

Define Team Types

- Stream-aligned Teams
- Enabling Teams
- Platform Teams
- Complicated Subsystem Teams



Align Teams with Domains

- Structure teams around Bounded Contexts from Domain-Driven Design

3

Define Team Types

- Stream-aligned Teams
- Enabling Teams
- Platform Teams
- Complicated Subsystem Teams

3

Define Team Types

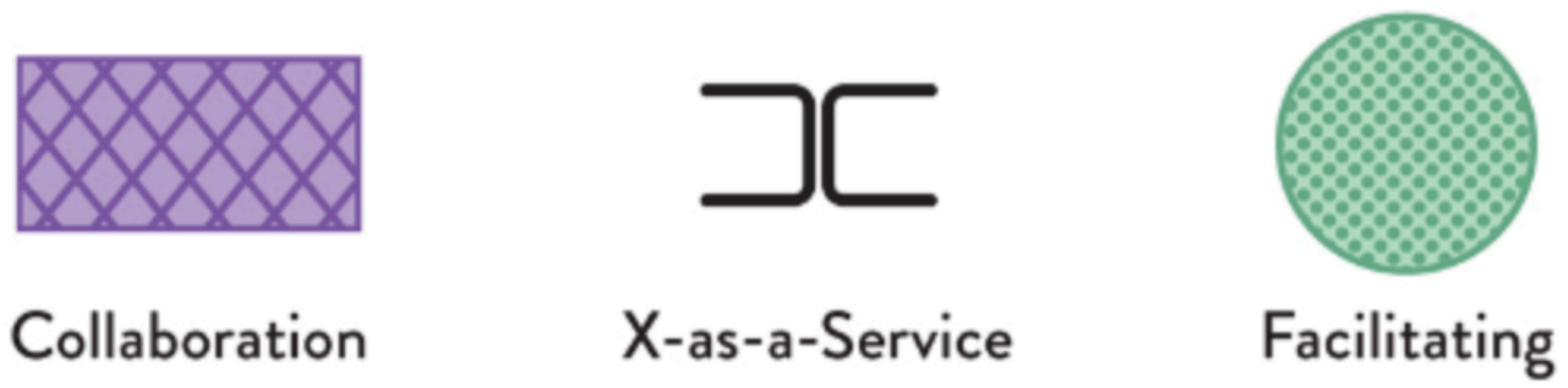
- ✤ Stream-aligned Teams
- ✤ Enabling Teams
- ✤ Platform Teams
- ✤ Complicated Subsystem Teams

Align Teams with Domains

- ✤ Structure teams around Bounded Contexts from Domain-Driven Design

Establish Interaction Mode Between Teams

- ✤ Collaboration
- ✤ Facilitating
- ✤ X-As-Service



Align Teams with Domains

- ✤ Structure teams around Bounded Contexts from Domain-Driven Design

Define Team Types

- ✤ Stream-aligned Teams
- ✤ Enabling Teams
- ✤ Platform Teams
- ✤ Complicated Subsystem Teams

Establish Interaction Mode Between Teams

- ✤ Collaboration
- ✤Facilitating
- ✤ X-As-Service

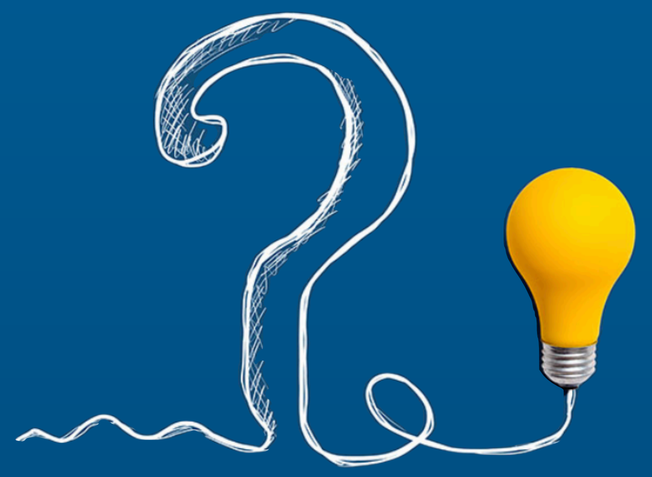
Minimise Cognitive Loads in Teams

Step 4

Finalise Transformation With Stabilisation & Continues Improvements

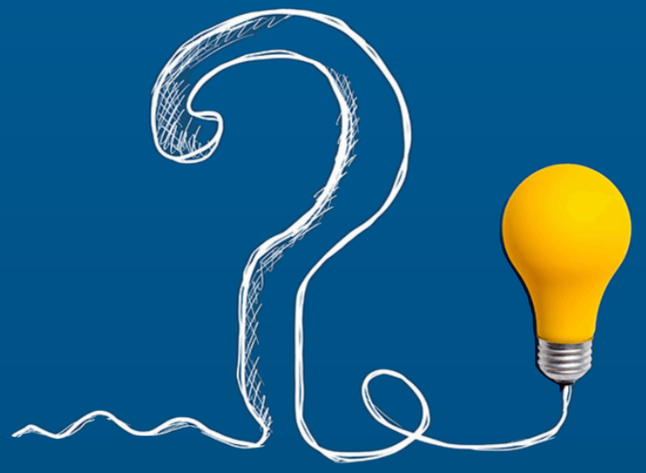
3

Leverage Systems Thinking

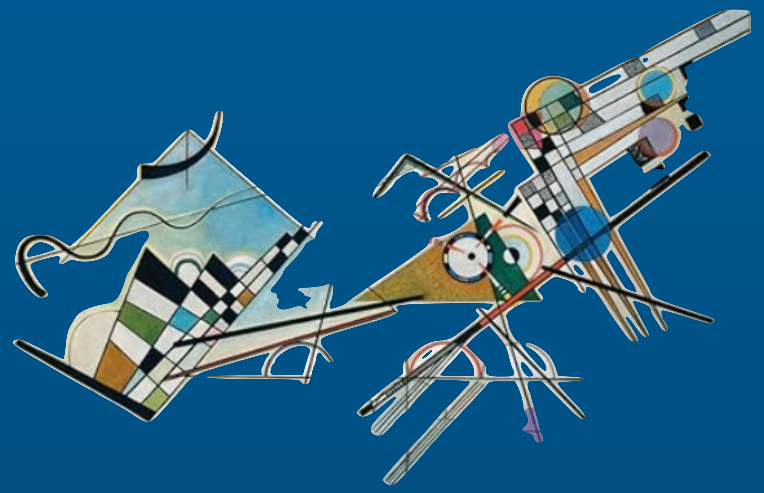


3

Leverage Systems Thinking

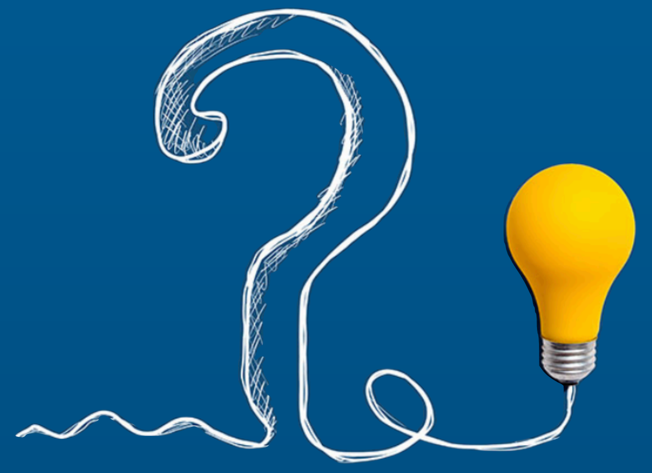


Refine DDD Models

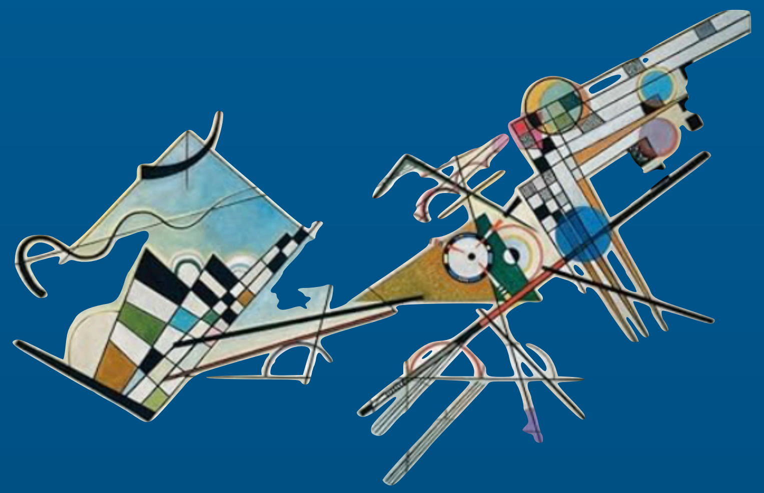


3

Leverage Systems Thinking



Refine DDD Models



Optimize Team Topologies

Thanks

Practical Envisioning of
Software Architecture

msd@chelongar.com

https://www.chelongar.com

Masoud Chelongar
Hands-on Software Architect