CIKLUM | SPEAKER'S CORNER

June, 04 | 17:00 (CET) | English

Anti-patterns in Enterprise Architecture

Oleksandr Savchenko

Solutioning Director
at Ciklum

Experiences of tomorrow. Engineered together.

CIKLUM

We transform how people experience the business. All through next generation technology.

What we do:

- Product Engineering
- Intelligent Automation
- Data & Analytics

2002 founded

4000+ professionals

20+ offices

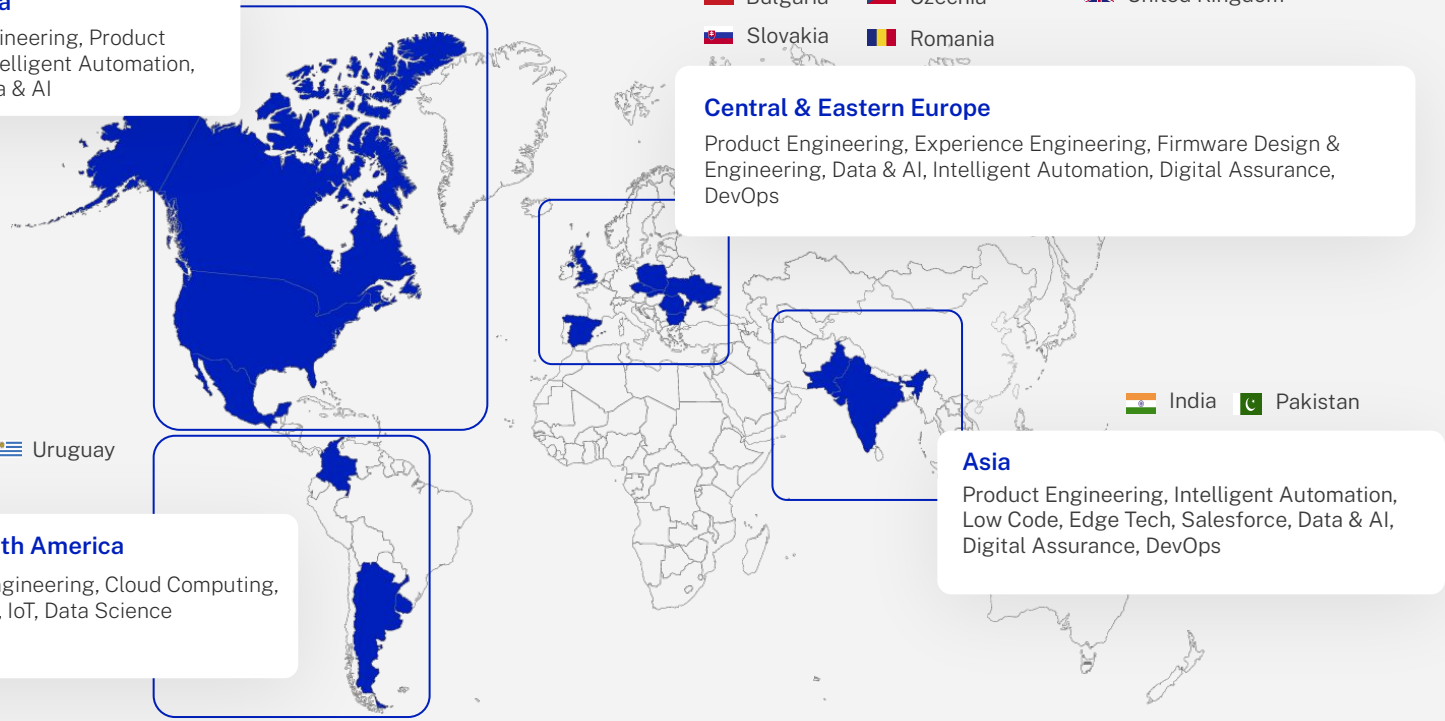
300+ clients

Leading companies choose us:

JUST EAT · Stena Line · eToro · betsson · AstraZeneca · TUI · KANTAR RETAIL · dacadoo

Our Global Expertise, to deliver Local Impact

Ciklum combines global reach with local insights to bridge the world's top technology talent and expertise



North America

Experience Engineering, Product Engineering, Intelligent Automation, Salesforce, Data & AI

Canada
USA
Mexico

Central & Eastern Europe

Product Engineering, Experience Engineering, Firmware Design & Engineering, Data & AI, Intelligent Automation, Digital Assurance, DevOps

Ukraine
Poland
Spain
Bulgaria
Czechia
United Kingdom
Slovakia
Romania

Asia

Product Engineering, Intelligent Automation, Low Code, Edge Tech, Salesforce, Data & AI, Digital Assurance, DevOps

India
Pakistan

South America

Experience Engineering, Cloud Computing, Cybersecurity, IoT, Data Science

Argentina
Uruguay
Columbia

CIKLUM

Meet the speaker

- 17+ years in IT
- Hands-on Enterprise Architect
- Led big programs (150+ engineers) and departments with 350+ engineers
- winner of Ukrainian IT Awards in category Software Engineering in 2019, Jury in 2020
- speaker on global conferences, author of courses

Oleksandr Savchenko
Solutioning Director, Ciklum

Agenda

CIKLUM

Patterns vs Anti-Patterns

Common terms and catalogues

Design by Committee

Business layer antipattern

Bad Data Virus

Data layer antipattern

Swiss Army Knife in the Distributed System

Application layer antipattern

Operational Over-Tooling

Technology layer antipattern

Useful materials

CIKLUM

Patterns vs Anti-Patterns

What is Architecture?

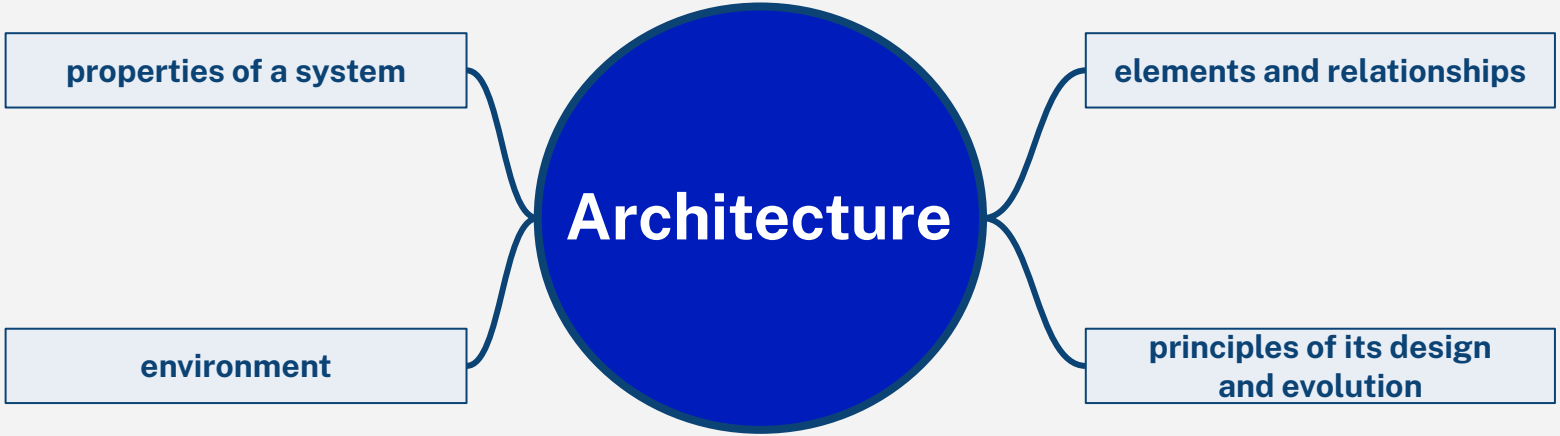
CIKLUM

ISO/IEC/IEEE 42010, Systems and software engineering - Architecture description

http://www.iso-architecture.org/42010/

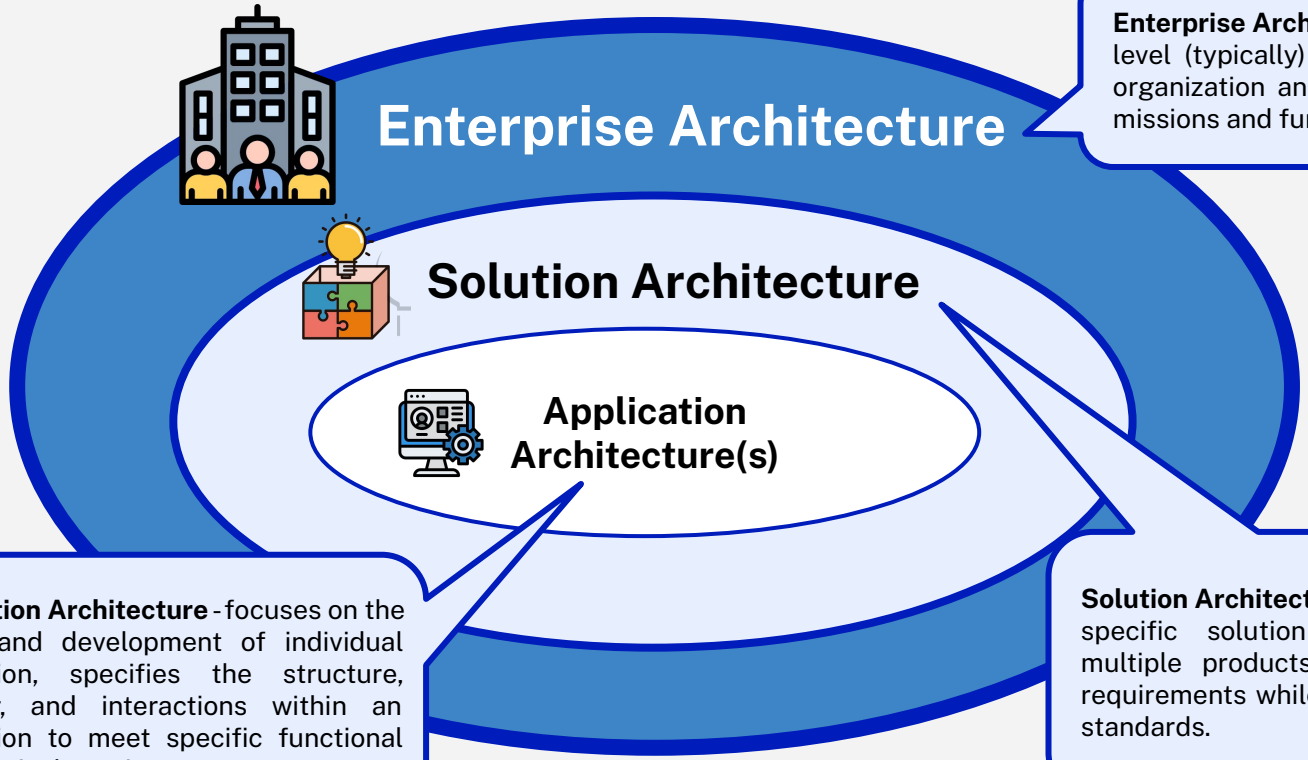
Architecture - is a fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution

http://www.iso-architecture.org/42010/defining-architecture.html



Architecture levels (ESA)

CIKLUM



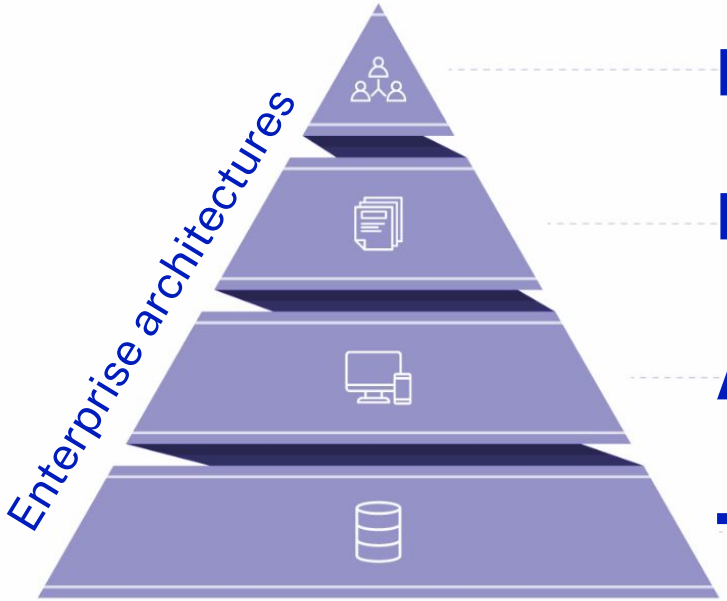
Enterprise Architecture - the highest level (typically) of description of an organization and typically covers all missions and functions.

Solution Architecture - defines how specific solution / product or multiple products meet business requirements while aligning with IT standards.

Application Architecture - focuses on the design and development of individual application, specifies the structure, behavior, and interactions within an application to meet specific functional and technical requirements.

Architecture levels (BDAT concept)

CIKLUM



B **Business** layer requires aligning technology and architecture decisions with business objectives and defines the business processes, strategies, and goals.

D **Data** layer is crucial for managing, storing, and securing data, and outlines how data flows and is structured across the organization.

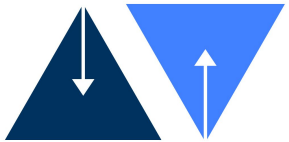
A **Application** layer concerns itself with software systems and their interactions, high-level application strategy and product standards.

T **Technology** layer encompasses the underlying infrastructure and platforms.

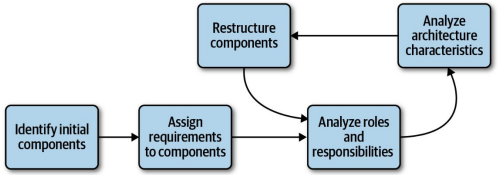
Federal Enterprise Architecture Framework - https://obamawhitehouse.archives.gov/sites/default/files/omb/assets/egov_docs/fea_v2.pdf

Architecture Methods and Frameworks

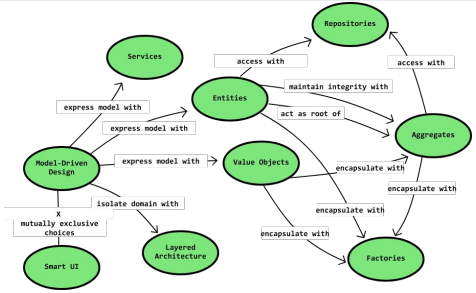
CIKLUM



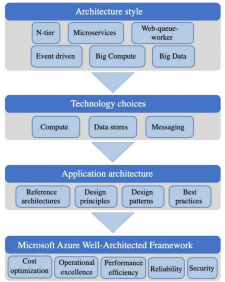
Top-to-Bottom, Bottom-Up approaches



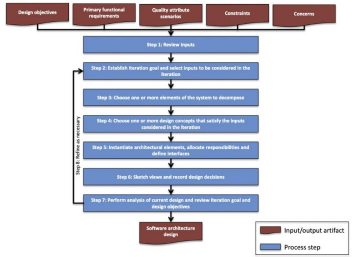
Component-Based Thinking

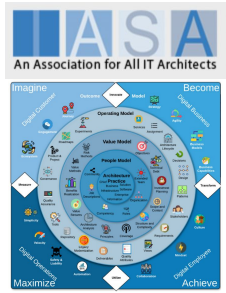


Domain Driven Design

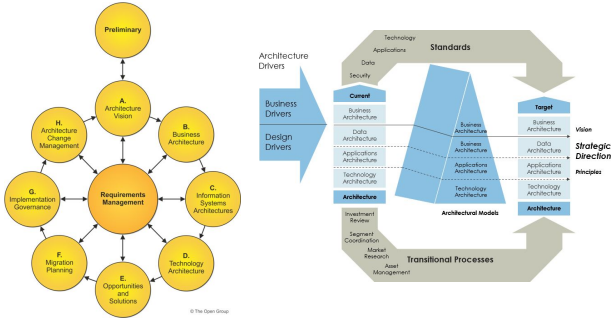


Architecture guidelines
(e.g. Microsoft Architecture guide, Azure/GCP/AWS/IBM Well-Architectured Framework, IBM Architectures, etc.)





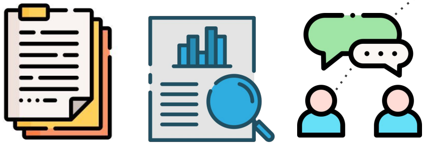
Architectural Methods
(SEI ADD, IASA)



Enterprise Architecture Frameworks
(e.g. TOGAF, Federal Enterprise Architecture Framework (FEAF), NATO Architecture Framework / NAF)

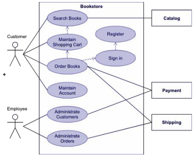
Simplified Architecture Development process

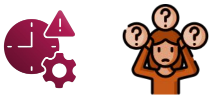
1 Collection of information from stakeholders via different elicitation methods



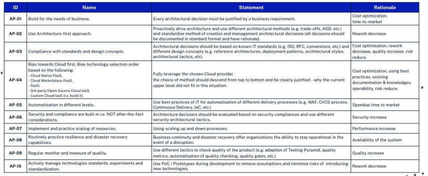
2 Architecturally Significant Requirements

Functional requirements, Constraints, Concerns, Quality Attribute Scenarios + Risks, Assumptions



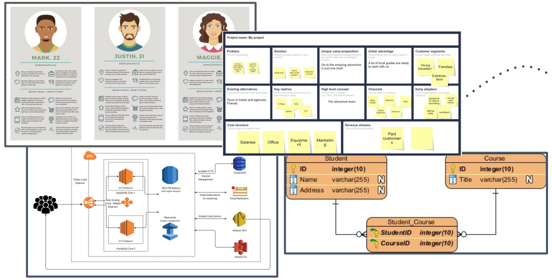


3 Architectural methods and tools, Design Principles, Architectural Tactics

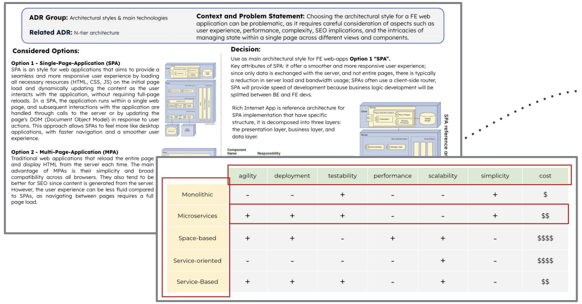


ID	Name	Statement	Rationale
AP-01	Build for the needs of business.	Every architectural decision must be justified by a business requirement.	Cost optimization, time-to-market
AP-02	Use Architecture-first approach.	Proactively drive architecture and use different architectural methods (e.g. trade-offs, ADD, etc.) and standardize method of creation and management architectural decisions (all decisions should be documented in standard format and have rationale).	Rework decrease
AP-03	Compliance with standards and design concepts.	Architectural decisions should be based on known IT standards (e.g. ISO, RFC, conventions, etc.) and different design concepts (e.g. reference architectures, deployment patterns, architectural styles, architectural tactics, etc).	Cost optimisation, rework decrease, quality increase, risk reduce
AP-04	Bias towards Cloud first. Bias technology selection order based on the following: - Cloud Native PaaS, - Cloud Marketplace PaaS, - SaaS, - 3rd-party/Open-Source Cloud IaaS, - Custom Cloud IaaS (i.e. build it).	Fully leverage the chosen Cloud provider. the choice of method should descend from top to bottom and be clearly justified - why the current upper level did not fit in this situation.	Cost optimization, using best practices, existing documentation & knowledges. operability, risk reduce.
AP-05	Automatisation in different levels.	Use best practices of IT for automatisation of different delivery processes (e.g. WAF, CI/CD process, Continuous Delivery, IaC, etc.)	Speedup time to market
AP-06	Security and compliance are built-in i.e. NOT after-the-fact considerations.	Architecture decisions should be evaluated based on security compliances and use different security architectural tactics.	Security increase
AP-07	Implement and practice scaling of resources.	Using scaling up and down processes.	Performance increase
AP-08	Routinely practice resilience and disaster recovery capabilities.	Business continuity and disaster recovery offer organizations the ability to stay operational in the event of a disruption.	Availability of the system
AP-09	Regular monitor and measure of quality.	Use different tactics to check quality of the product (e.g. adaption of Testing Pyramid, quality metrics, automatisation of quality checking, quality gates, etc.)	Quality increase
AP-10	Actively manage technologies standards, experiments and standardization.	Use PoC / Prototypes during development to remove assumptions and minimize risks of introducing new technologies.	Rework decrease

4 Architectural Views



5 Architectural Decisions



ADR Group: Architectural styles & main technologies

Related ADR: N-tier architecture

Context and Problem Statement: Choosing the architectural style for a FE web application can be problematic, as it requires careful consideration of aspects such as user experience, performance, complexity, SEO implications, and the intricacies of managing state within a single page across different views and components.

Considered Options:

Option 1 - Single-Page-Application (SPA)
SPA is an style for web applications that aims to provide a seamless and more responsive user experience by loading all necessary resources (HTML, CSS, JS) on the initial page load and dynamically updating the content as the user interacts with the application, without requiring full-page reloads. In a SPA, the application runs within a single web page, and subsequent interactions with the application are handled through calls to the server or by updating the page's DOM (Document Object Model) in response to user actions. This approach allows SPAs to feel more like desktop applications, with faster navigation and a smoother user experience.

Option 2 - Multi-Page-Application (MPA)
Traditional web applications that reload the entire page and display HTML from the server each time. The main advantage of MPAs is their simplicity and broad compatibility across all browsers. They also tend to be better for SEO since content is generated from the server. However, the user experience can be less fluid compared to SPAs, as navigating between pages requires a full page load.

Decision:

Use as main architectural style for FE web-apps **Option 1 "SPA".**
Key attributes of SPA: it offer a smoother and more responsive user experience; since only data is exchanged with the server, and not entire pages, there is typically a reduction in server load and bandwidth usage; SPAs often use a client-side router, SPA will provide speed of development because business logic development will be splitted between BE and FE devs.

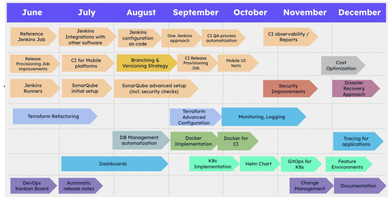
Rich Internet App is reference architecture for SPA implementation that have specific structure, it is decomposed into three layers: the presentation layer, business layer, and data layer.

SPA reference ar

Component Name Responsibility

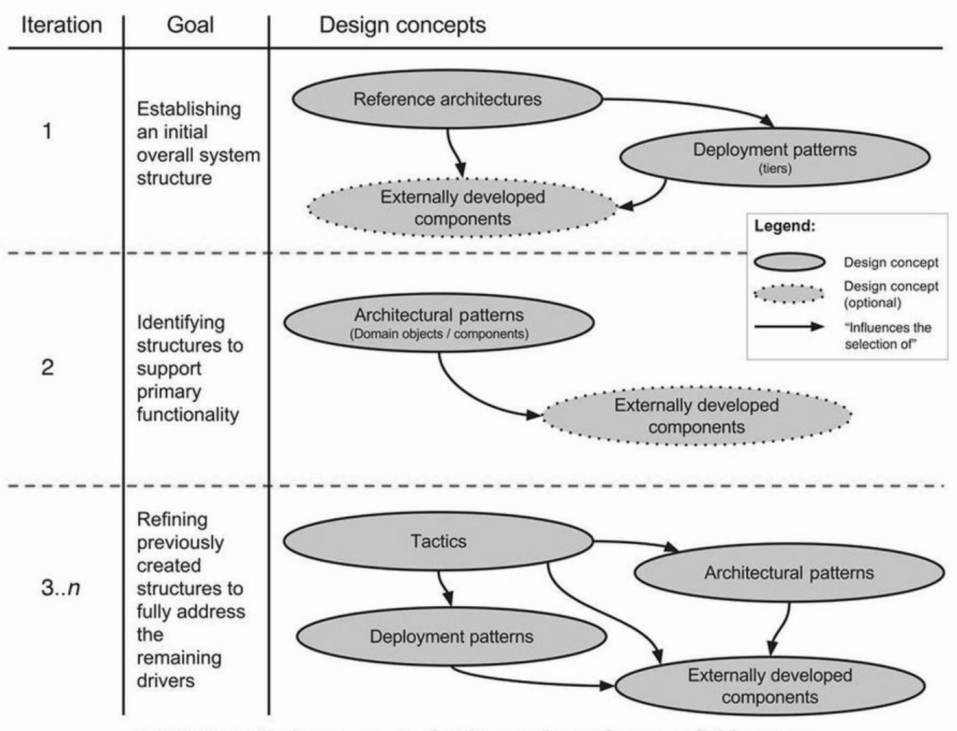
	agility	deployment	testability	performance	scalability	simplicity	cost
Monolithic	-	-	+	-	-	+	$
Microservices	+	+	+	-	-	+	$$
Space-based	+	+	-	+	+	-	$$$$
Service-oriented	-	-	-	-	+	-	$$$$
Service-Based	+	+	+	-	+	-	$$

6 Implementation plan with evolutionary approach



Architecture Design Concepts

Usually Architectural Design is iterative process where You should use multiple Design Concepts to speed up process of architecture creation and be aligned with IT standards and best practices.

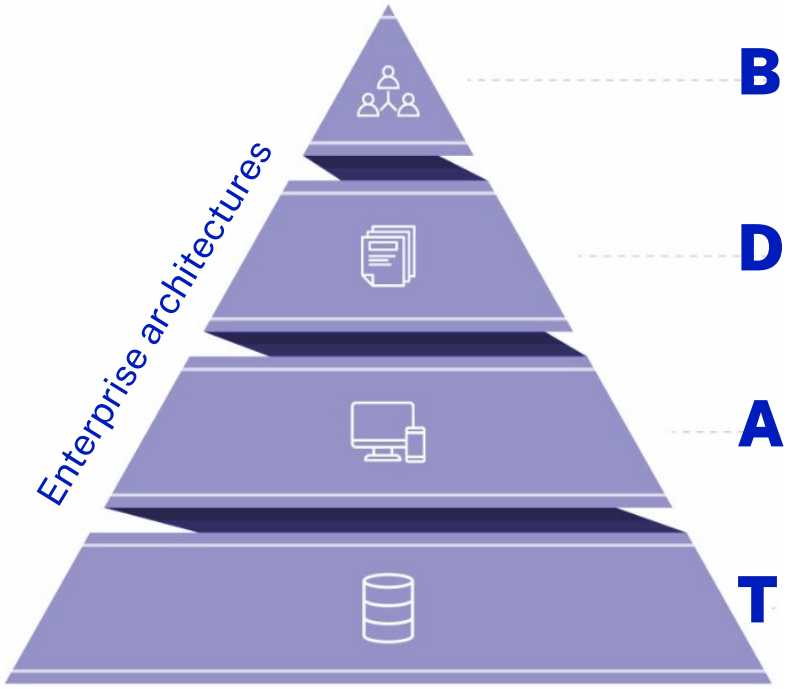


DESIGN CONCEPTS:

- Reference Architectures
- Architectural Tactics
- Deployment Patterns
- Standards (e.g. RFC, ISO)
- Tools, Dev Frameworks, Platforms, Technologies
- **Architectural Patterns / Styles**

Enterprise Architecture Patterns and Styles

CIKLUM



Business Architecture:
Capability-Based Planning patterns, Customer Journey-Based Architecture, Value Stream Mapping Pattern, Organization & Role-Based Patterns (Organization Units, Roles, RACI matrices, Enterprise Operating Models), Business Motivation Model (BMM) Pattern, Business Process-Oriented Patterns, Policy and Rules-Based Patterns

Data Architecture:
Data Warehouse, Data Lake, ETL/ELT, Change Data Capture, Data Mesh, Data Virtualization, Data Streaming, Lambda / Kappa Architecture, Big-Data, Centralized vs Distributed Data, Federated Data, Domain-Oriented Data Ownership, Transactional vs. Analytical, Real-Time vs. Batch Processing

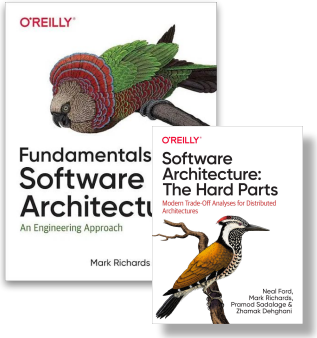
Application Architecture:
Layered, Hexagonal (Ports and Adapters), Microservices, Monolithic, Modular Monolith, Service-Based Architecture, SOA, Event-Driven, CQRS, DDD, BFF, API Gateway, Saga, Strangler Fig Pattern, Serverless, Function as a Service (FaaS), Reactive Architecture, Actor Model, Micro-Frontend.

Technology Architecture:
N-Tier Architecture, Public / Private Cloud, Hybrid/Multi-Cloud, Edge Computing, Containerization, Infrastructure as Code (IaC), Service Mesh, Zero Trust Architecture, Platform Engineering, Observability-Driven Architecture, Resilient Architecture, X-Y-Z scaling and availability cube, Immutable Infrastructure, Failover Cluster.

Where can you find catalogs?

This is known for architectural patterns and styles

CIKLUM



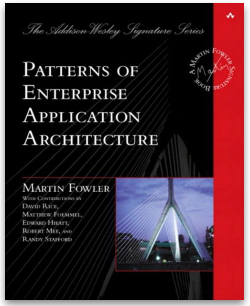
Azure Architecture Center

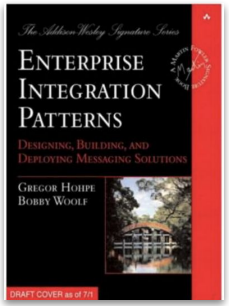
Guidance for architecting solutions on Azure using established patterns and practices

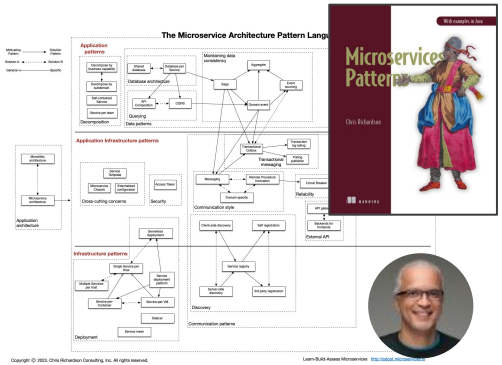
Data Management

Messaging

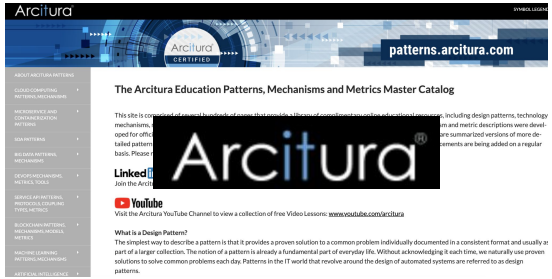
Design and Implementation







https://microservices.io/



https://patterns.arcitura.com/

What is an Anti-Pattern?

CIKLUM

Neal Ford

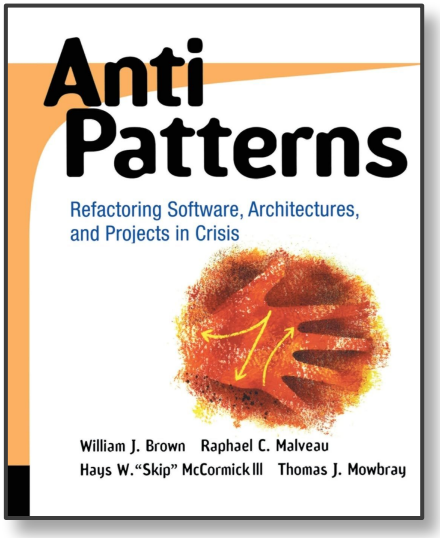
*"**Antipattern** is a practice that **initially** looks like a **good idea,** but turns out to be a **mistake** … and **better alternatives exist** …"*

---

*"**Pitfall** looks superficially like a good idea but immediately reveals itself to be a **bad path…**"*

Book/Catalogs of Anti-Patterns

CIKLUM



"AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis"

Publication date: 1998

http://antipatterns.com/

41 Antipatterns and Mini-Antipatterns

Architecture

1. Autogenerated Stovepipe
2. Stovepipe Enterprise
3. Jumble
4. Stovepipe System
5. Cover Your Assets
6. Vendor Lock-In
7. Wolf Ticket
8. Architecture By Implication
9. Warm Bodies
10. Design By Committee
11. Swiss Army Knife
12. Reinvent the Wheel
13. The Grand Old Duke of York

Development

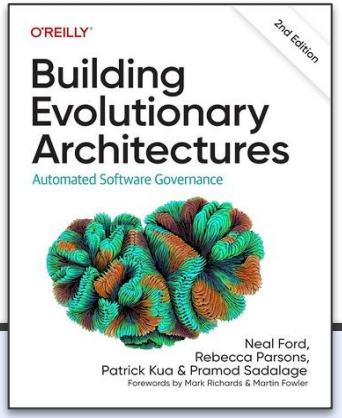
1. The Blob
2. Continuous Obsolescence
3. Lava Flow
4. Ambiguous Viewpoint
5. Functional Decomposition
6. Poltergeist
7. Boat Anchor
8. Golden Hammer
9. Dead End
10. Spaghetti Code
11. Input Kludge
12. Walking through a Minefield
13. Cut-and-Paste Programming
14. Mushroom Management

Project Management

1. Blowhard Jamboree
2. Analysis Paralysis
3. Viewgraph Engineering
4. Death by Planning
5. Fear of Success
6. Corncob
7. Intellectual Violence
8. Irrational Management
9. Smoke and Mirrors
10. Project MisManagement
11. Throw It over the Wall
12. Fire Drill
13. The Freud
14. E-mail Is Dangerous

Books/Catalogs of Anti-Patterns

CIKLUM



Technical Architecture:

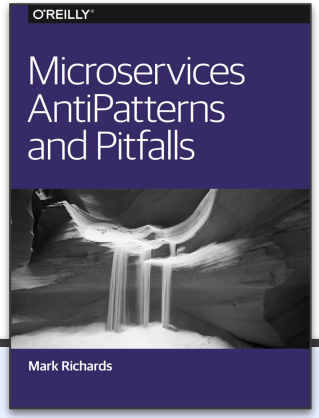
1. Antipattern: Last 10% Trap and Low Code/No Code
2. Antipattern: Vendor King
3. Pitfall: Leaky Abstractions
4. Pitfall: Resume-Driven Development

Incremental Change:

1. Antipattern: Inappropriate Governance
2. Pitfall: Lack of Speed to Release

Business Concerns:

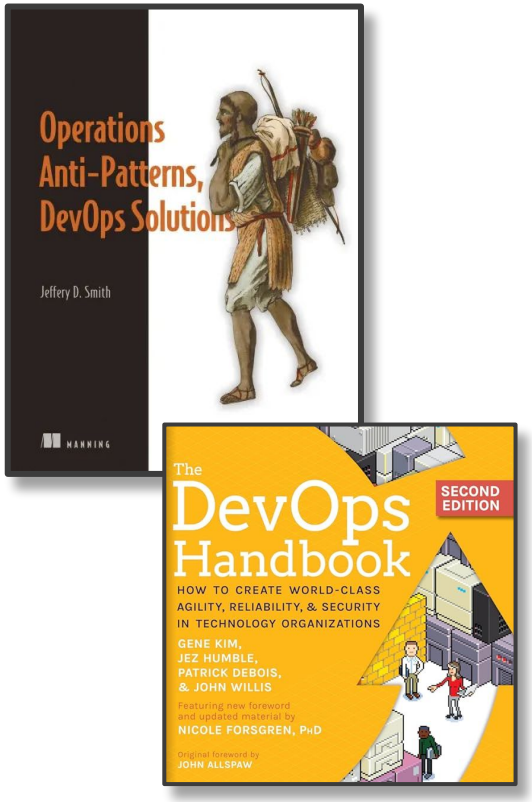
1. Pitfall: Product Customization
2. Antipattern: Reporting Atop the System of Record
3. Pitfall: Excessively Long Planning Horizons



1. Data-Driven Migration AntiPattern
2. The Timeout AntiPattern
3. The “I Was Taught to Share” AntiPattern
4. Reach-in Reporting AntiPattern
5. Grains of Sand Pitfall
6. Developer Without a Cause Pitfall
7. Jump on the Bandwagon Pitfall
8. The Static Contract Pitfall
9. Are We There Yet Pitfall

Catalogs of DevOps Anti-Patterns

CIKLUM



https://web.devopstopologies.com

Anti-Type A: Dev and Ops Silos
Anti-Type B: DevOps Team Silo
Anti-Type C: Dev Don't Need Ops
Anti-Type D: DevOps as Tools Team
Anti-Type E: Rebranded SysAdmin
Anti-Type F: Ops Embedded in Dev Team
Anti-Type G: Dev and DBA Silos
Anti-Type H: Fake SRE

AWS Well-Architected

DevOps Guidance

116 antipattern overviews
within **26** practices
around **5** categories
(Organizational adoption, Development lifecycle, Quality Assurance, Automated Governance, Obiervability)

Agenda for today

CIKLUM

B – **Design by Committee –** Decisions are made by large, misaligned stakeholders to bloated, slow-moving architectures that reflect compromise over clarity, often lacking clear ownership or vision.

D – **Bad Data Virus –** Poor-quality or inconsistent legacy data spreads through systems like a virus, and this data included in analytics, automation, and downstream services, making recovery increasingly difficult over time.

A – **Swiss Army Knife –** components are over-engineered with too many features or responsibilities, becoming inflexible, difficult to maintain, and ultimately failing to serve any one purpose well.

T – **Operational Over-Tooling –** The tech stack becomes cluttered with overlapping tools for monitoring, deployment, and ops, creating complexity, skill silos, and increased maintenance overhead without proportional benefit.

CIKLUM

Business
Data
Application
Technology

Antipattern: Design by Committee

Antipattern: Design by Committee

The main problems we usually face

CIKLUM

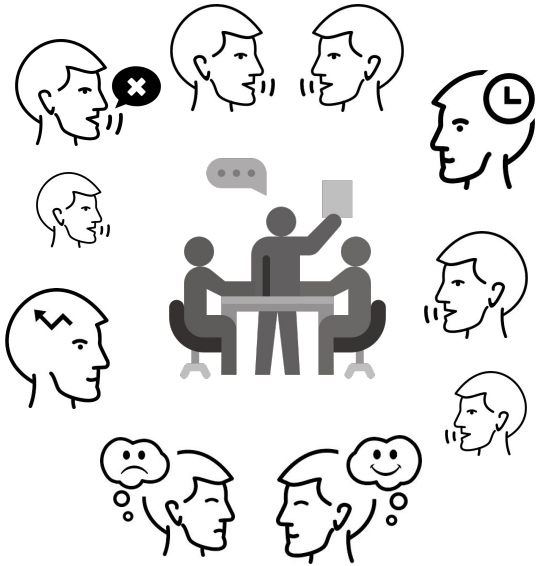
Architecture is not presented or not understandable → **Re-architecture** during the development → Overcomplicated designs and continuous refactoring → Cost for maintenance and extensibility

No clear architectural decision-making process → Many people want to influence the design → Long/Hard meetings with conflicts, rescheduling, without outcomes → Slow delivery and increase time-to-market

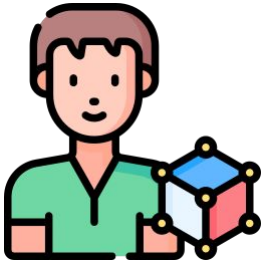
No consistent vision of how to measure success (quantitative metrics) → Hard and long process for release preparation → Lack of Observability after Release → Released not relevant product, Business doesn't meet KPIs

Antipattern: Design by Committee

Let’s try to solve these problems…



Collective Design

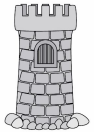


Dedicated Architect(s)

Potential Pitfalls / Antipatterns

Architect play Golf

Architects do not participate in the project after the architecture phase is done and expect strict compliance of development with the design



Ivory Tower Architect

Architects design and processes without sufficient input from developers or operational teams, leading to impractical or overly complex architectures.

Antipattern: Design by Committee

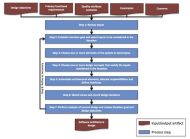
Let’s try to solve these problems...

CIKLUM

Technical / Design Committee

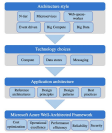
Architect(s), Tech Lead(s), QA Lead, DevOps Lead, Delivery Lead, Product Lead

1 Choose Architectural Frameworks, Methods



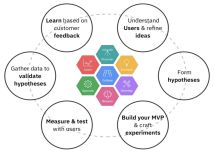
SEI ADD

Top-to-Bottom approach



MS architecture guide

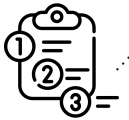
Trade-offs



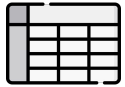
Hypothesis Driven (PoC & Prototyping)

2 Specify clear Architecture Decision Making process

1. Initial Requirements



2. ASRs

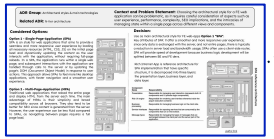


Functional requirements, Constraints, Concerns, Quality Attribute Scenarios

3. Architectural principles

4. Solutions Options, Trade-Off

5. ADRs



6. Implementation plan

Antipattern: Design by Committee

Let's try to solve these problems...

CIKLUM

Technical / Design Committee

Architect(s), Tech Lead(s), QA Lead, DevOps Lead, Delivery Lead, Product Lead

3 Choose Architectural Viewpoints and Tools

Choosed viewpoints:

UNIFIED MODELING LANGUAGE UML

C4 model

Tools for creation diagrams:

PlantUML

draw.io

List of main diagrams:

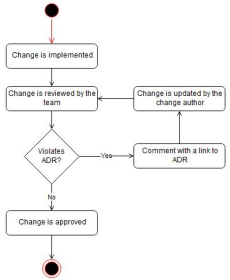
- System context
- Container
- Detailed Component
- Deployment
- Auth sequence
- Main data flow activity
- Physical ERD
- ...

4 Implement ADRs and select reference format

ADR Format (attributes):

- Id
- Title
- Status
- Related ADRs
- Group
- Context and Problem Statement
- Considered Options
- Decision

Creation and approvals Process of ADRs

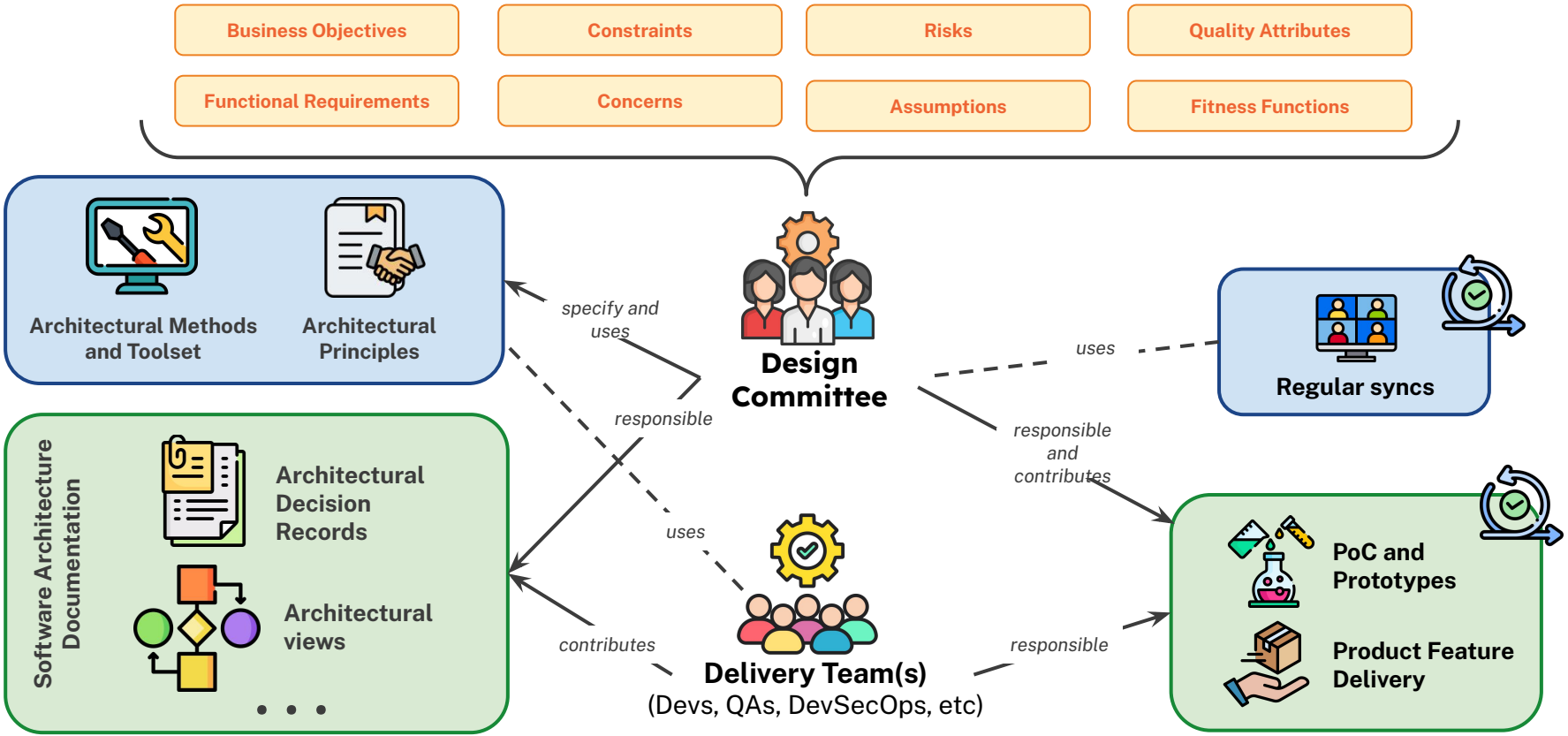


Example AWS Guide - ADR process

Antipattern: Design by Committee

CIKLUM

... and finally present Architectural methods and provide possibility for contribution



Antipattern: Design by Committee

CIKLUM

Measurable Metrics to support in identification of this anti-pattern

Collaboration

Duration of Meetings,

Number of Decision-Makers Involved,

Time to Decision (from Idea to Decision),

Team Satisfaction NPS (Net Promoter Score) by Survey

Design Consistency

Current vs Target Architecture (# of components/modules per complexity),

# of redundant components or services that provide overlapping functionality,

Architecture Document Change Frequency

Complexity and Debt

Code Complexity Metrics,

Total technical debt (in person-hours or cost) / Total development effort,

Number of defects per KLOC (thousands of lines of code),

Integration Failure Rate

Project Delay and Timelines

Number of design changes per sprint or milestone or release (e.g. use Jira Labels to mark),

Schedule Variance - Actual vs Planned Timelines,

Feature Lead Time (feature from concept to production),

Team Velocity, Release Burndown Rate

Cost and Productivity

Refactoring effort,

Percentage of budget allocated to rework vs original design,

Ratio of productive time to total time (including meetings and rework),

Spikes vs Features per Sprint

Antipattern: Design by Committee

CIKLUM

How to mitigate or fix this

1. **Allocate Architect:** Ensure a strong leader is responsible for the final decisions.
2. **Create Technical / Design Committee with minimum final Decision-Makers:** Implement decision-making frameworks that streamline the process, such as the RACI to clarify clear roles and responsibilities.
3. **Unify Framework for Analyses Requirements:** Create a clear flow on how to identify ASRs (business objectives, constraints. concerns, quality attributes, ...) for all working item (big PI and release, Feature, User Story)
4. **Implement Fitness Functions:** Regularly measure the architecture's alignment with its intended design principles using fitness functions. Automate these checks where possible to receive continuous feedback.
5. **Specify Architectural Principles:** Stick to core design principles and avoid making changes that deviate from the established goals and guidelines.
6. **Use Architecture Decision Records (ADRs):** Document every major architectural decision, including the rationale, alternatives considered, and the final decision. This transparency reduces redundant discussions and ensures alignment.
7. **Promote a Unified Vision:** Ensure that all stakeholders understand and buy into a clear architectural vision and set of principles. Workshops, shared documentation, and collaborative sessions can help align the team.
8. **Conduct Regular Architectural Reviews:** Perform regular reviews focused on alignment, consistency, and adherence to the architectural vision. Use metrics and fitness functions to guide these reviews.
9. **Prioritize Feedbacks:** Gather input from all stakeholders, but prioritize and implement feedback that aligns with the project's vision and goals.

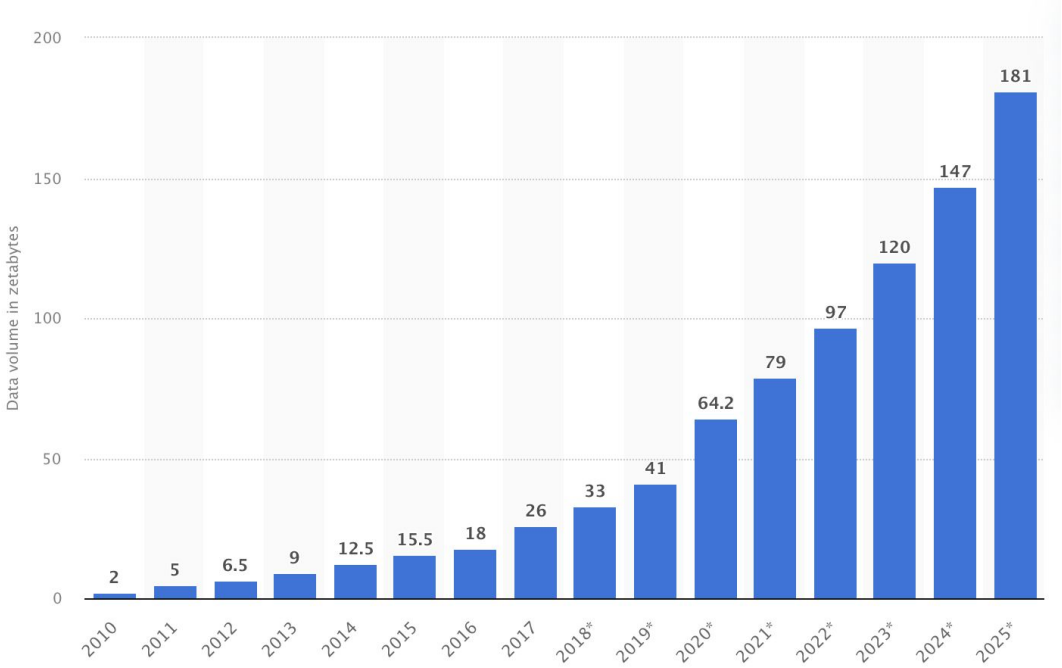
CIKLUM

Business
Data
Application
Technology

Antipattern: Bad Data Virus

Antipattern: Bad Data Virus

Main problem: Growth of Data



Forecast: 175 ZB will be created by 2025

https://www.datanami.com/2022/01/11/big-growth-forecasted-for-big-data/

According to the latest estimates -
402.45 million terabytes of data are created **each day**

~2.8 zettabytes per week
~12 zettabytes per month
~147 zettabytes per year

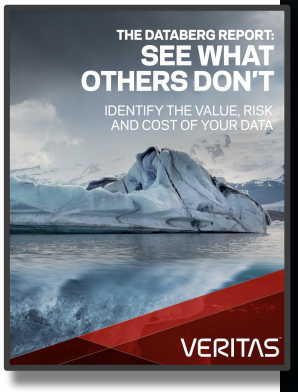
Antipattern: Bad Data Virus

Interesting point that most of data is Dark Data

> *"**Dark Data** is information assets organisations collect, process and store during regular business activities, but generally fail to use for other purposes."*
>
> *Gartner, Inc.*
>
> It means "collected, but not used"

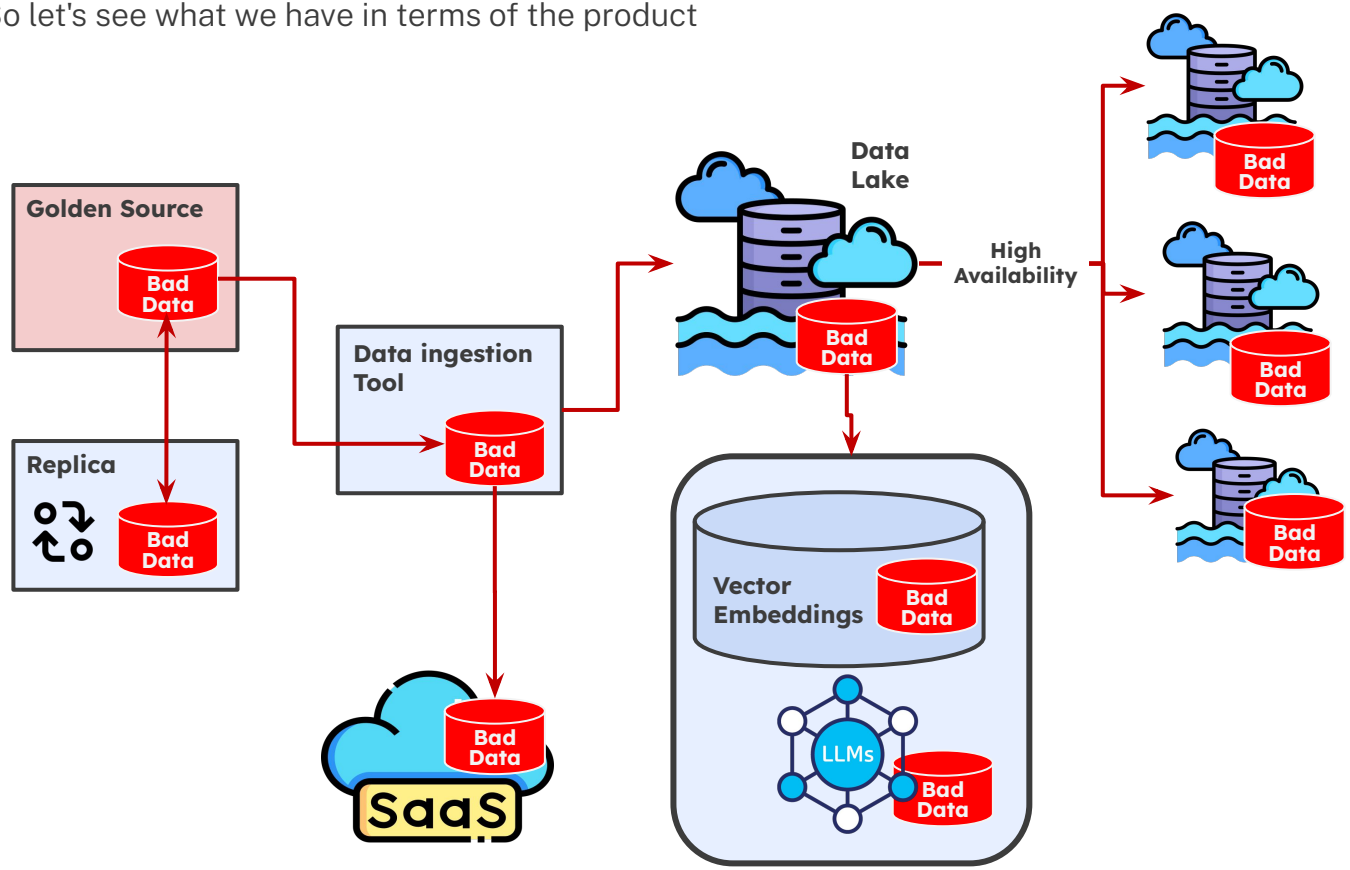
Report by VERITAS

to explain 3 types of data:
- Business critical data
- Rot data
- Dark data



Antipattern: Bad Data Virus

So let's see what we have in terms of the product



Fear Of Deleting Data

Antipattern: Bad Data Virus

CIKLUM

Measurable Metrics to support in identification of this anti-pattern

1. Data Usage

Data Access Frequency,
Data Read/Write Ratio,
Query Patterns and Complexity,
Data Retrieval Latency,
Data Deletion and Purging Rate,
User Query and Access Logs,
User Feedback and Usage Surveys

2. Data Storage

Data Volume and Growth Rate,

Storage Utilization Rate,

Data Redundancy Ratio,

Archival vs. Active Data Ratio,

Data Age Distribution,

Data Lifecycle Stages

3. Data Quality

Data Completeness - assesses whether all required data elements are present,

Data Accuracy - how well the data reflects real-world values or facts,

Data Freshness and Staleness - how up-to-date data is,

4. Data Governance and Compliance

Data Retention metrics,

Access Control and Data Sensitivity,

Compliance Audit Results

5. Performance and Efficiency

Cache Hit/Miss Ratio,

System Resource Utilization for Data Operations,

Data Transfer Volume and Bandwidth Usage

6. Observability and Monitoring

Data Pipeline Metrics - health and efficiency of data pipelines,

Log Data Utilization - unused log data is a common form of dark data,

Anomaly Detection in Data Usage patterns - identifies unusual patterns in data usage

Antipattern: Bad Data Virus

How to mitigate or fix this

CIKLUM

Clear Data Governance Strategy

1 Conduct a Data Audit

2 Implement a data classification approach

3 Develop a data retention / archiving / deletion policies

4 Automate data management processes

CIKLUM

Business
Data
Application
Technology

Antipattern: Swiss Army Knife

Antipattern: Swiss Army Knife in distributed systems

CIKLUM

"Swiss Army Knife" is a classical Anti-Pattern in the Software Development

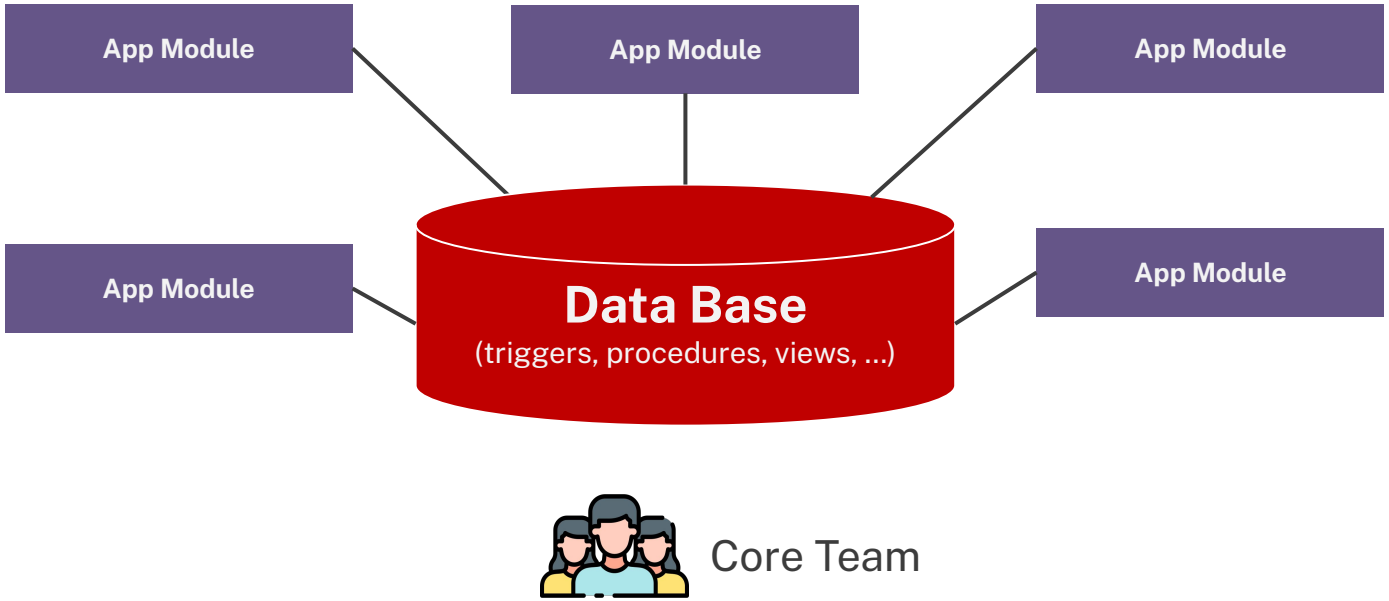
is a tool with so many features

A Swiss Army Knife, is an excessively **complex class, interface**.

Architect attempts to provide for **all possible uses** of the class and class may include from dozens to thousands of method signatures for a single class.

Antipattern: Swiss Army Knife in distributed systems

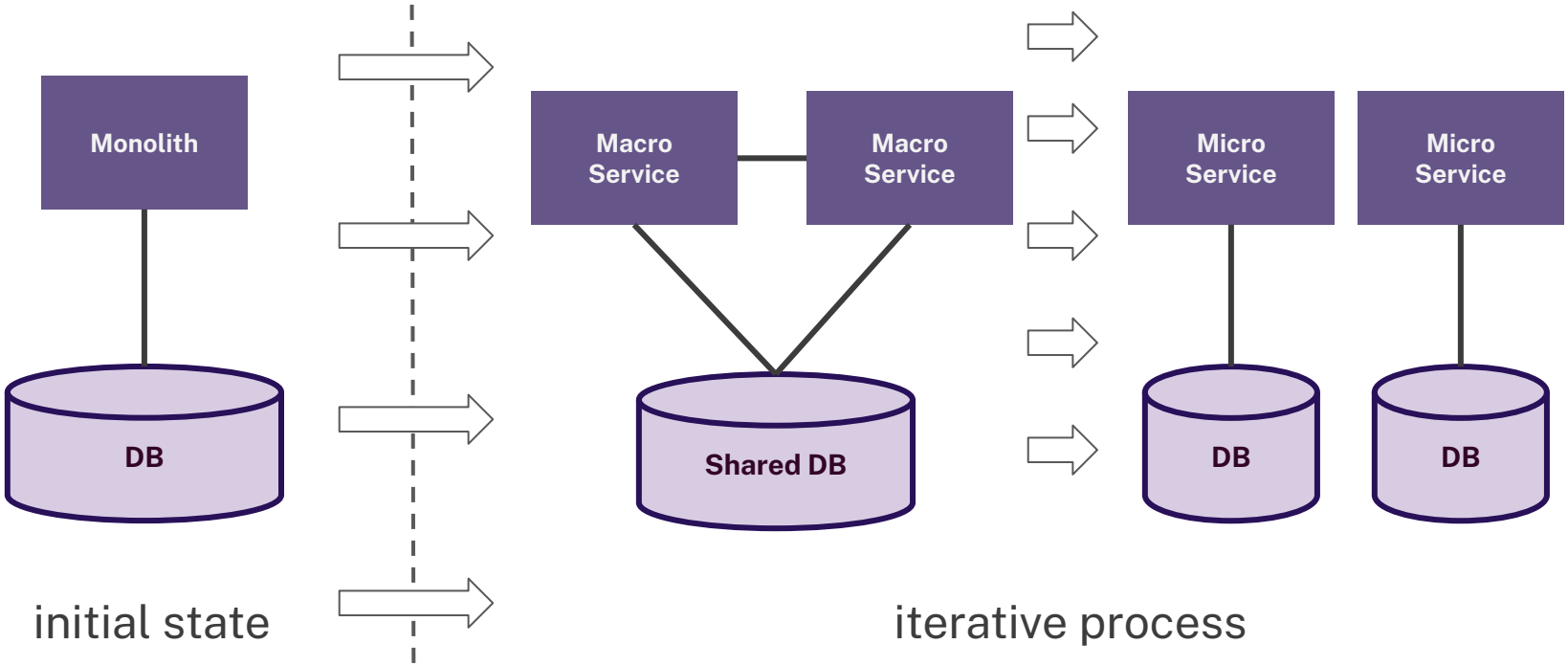
Let's back to 2000 ... 2010 ...



Antipattern: Swiss Army Knife in distributed systems

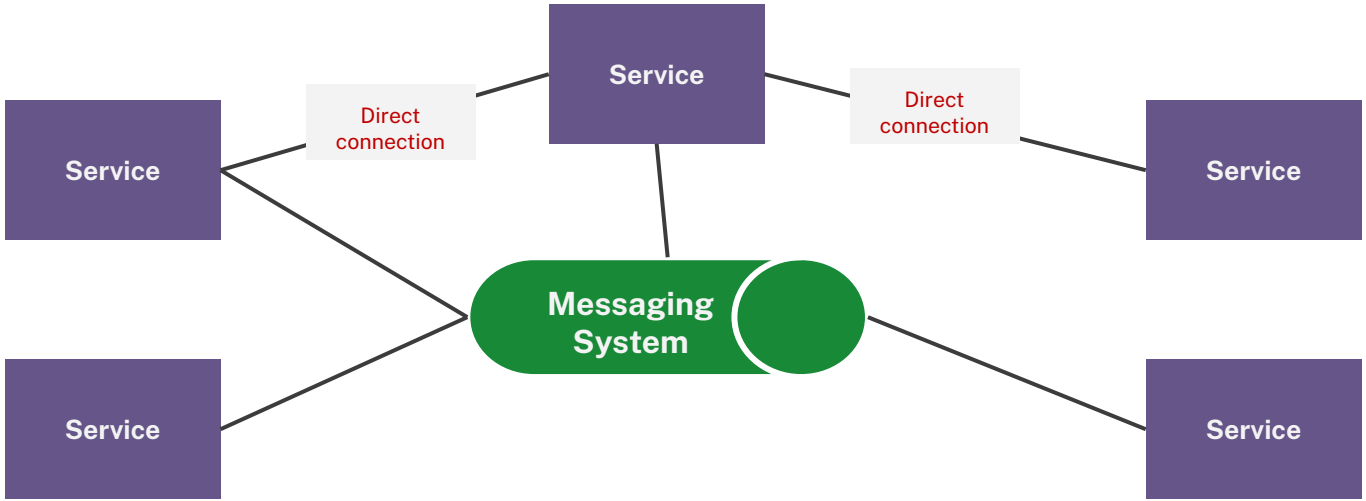
CIKLUM

... in 2010+ we had a classical case “migration to services architecture”



Antipattern: Swiss Army Knife in distributed systems

Classical situation - migration to services architecture



1. Adding Messaging system for service communication

2. Orchestration or Choreography dilemma

3. Decoupling Business Logic from Services

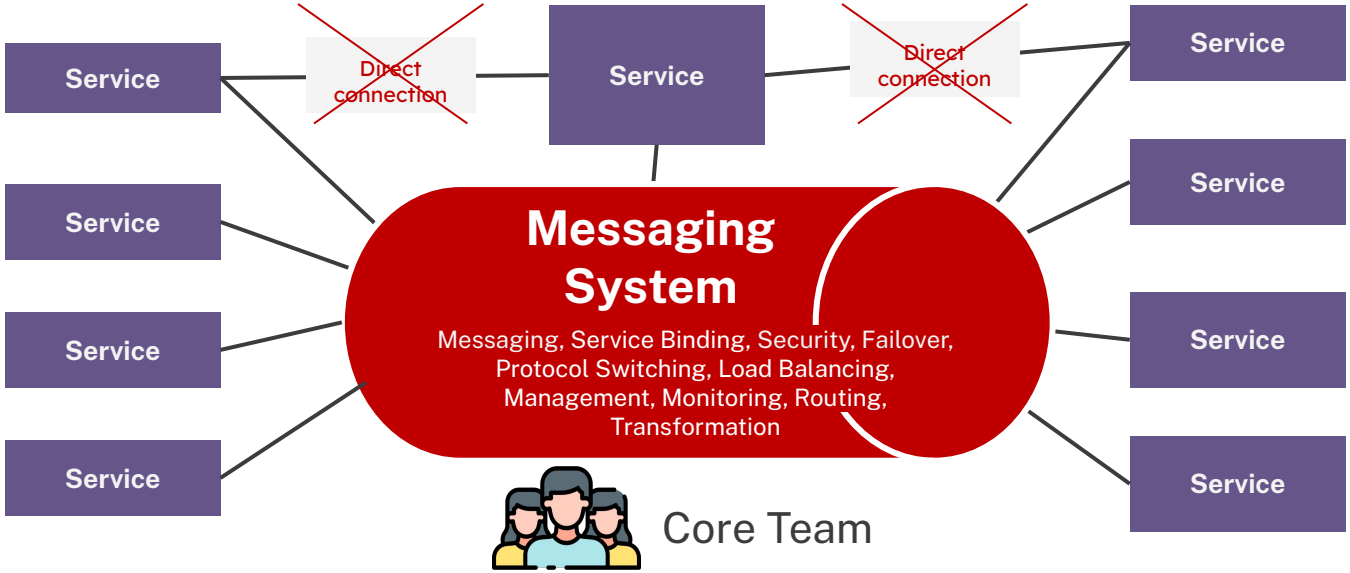
4. Add more responsibility to messaging system (aka ESB)

5. Dedicated Team to support

Antipattern: Swiss Army Knife in distributed systems

CIKLUM

Classical situation - migration to services architecture



1. Adding Messaging system for service communication

2. Orchestration or Choreography dilemma

3. Decoupling Business Logic from Services

4. Add more responsibility to messaging system (aka ESB)

5. Dedicated Team to support

Antipattern: Swiss Army Knife in distributed systems

CIKLUM

Main symptoms

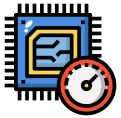
- **Overloaded Messaging System**: doing too much - handling everything from routing to orchestration to business logic.
- **Performance Bottlenecks:** The ESB may become a performance bottleneck if it's not properly scaled, impacting the overall responsiveness.
- **Difficulty in Making Changes:** Difficulty in making changes or upgrades to the messaging system without affecting multiple services.
- **Difficulty in Debugging and Monitoring**: Complex message flows make it difficult to trace, debug, and monitor across services.
- **Hidden Business Logic**: Core business logic is hidden inside message routing or processing rules, making it hard to understand system behavior.
- **Tight Coupling via Message Context**: services depend on specific message formats, leading to tight coupling and cascading changes.

https://developer.ibm.com/articles/cl-lightweight-integration-1/

Antipattern: Swiss Army Knife in distributed systems

CIKLUM

Measurable Metrics to support in identification of anti-pattern



1. Messaging System Utilization

CPU and Memory Usage,
Throughput (Messages per Second),
Latency per Message Processing

2. Number of Message Types and Transformations

Total # of Message Types,
Number of Transformations per Message

3. Services Coupling

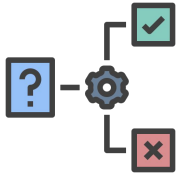
of Services dependent on Specific Message Types,
Changes in Upstream Services Affecting Downstream Services

4. Failure Propagation

Impact Radius of Messaging System Failures,
Mean Time to Detect (MTTD) and Mean Time to Recover (MTTR) from Message Failures

5. Monitoring and Debugging

Time to Trace a Transaction Across Services,
Percentage of Messages with Complete Tracing and Logging Information



6. Business Logic in Messaging Layer

Percentage of Business Logic Implemented in the Messaging Layer

Antipattern: Swiss Army Knife in distributed systems

CIKLUM

How to mitigate or fix this

1. **Refactor to simplify the Messaging System:**
   - Review and remove complex processing rules, scripts, or logic embedded in the messaging system.
   - Use lightweight message systems (e.g., Apache Kafka, RabbitMQ) without complex processing.
   - For cases requiring more advanced orchestration, consider dedicated workflow engines (e.g., Apache Airflow, Cadence, or Camunda) instead of the message broker.

2. **Simplify Message Types and Minimize Transformations** - each message should have a clear purpose and format
   - Audit current message types and transformations to identify and remove unnecessary ones.
   - Use simpler, more generic message formats that contain only essential information, avoiding excessive nesting or deep hierarchies.
   - Minimize the need for data transformation by standardizing communication formats (e.g., JSON, Avro, Protocol Buffers).

3. **Establish clear Messaging Protocols and Contracts:**
   - Use schema registries (e.g., Confluent Schema Registry) or API specifications (e.g., OpenAPI, AsyncAPI) to define and enforce message formats.
   - Implement consumer-driven contract testing (e.g., Pact) to ensure that changes in message formats do not break downstream consumers.
   - Version messages and provide backward compatibility for updates to ensure that changes do not disrupt the entire system.

4. **Decentralize Business Logic to Microservices:**
   - Identify business logic currently executed in the messaging layer. Refactor these operations into stateless or stateful microservices where they naturally belong.
   - Ensure that microservices expose clear APIs and endpoints for business operations, rather than relying on message-driven choreography.
   - Use asynchronous messaging for triggering actions without embedding complex business rules in message flows.

5. **Adopt a "Smart Endpoints, Dumb Pipes" Strategy (decision-making within services themselves)**

6. **Improve Observability and Monitoring of Messaging Flows (e.g. distributed tracing)**

7. **Implement Domain-Driven Design (DDD) Principles (e.g. domain events for communication between services)**

8. **Conduct Regular Architecture Reviews and Refactoring**

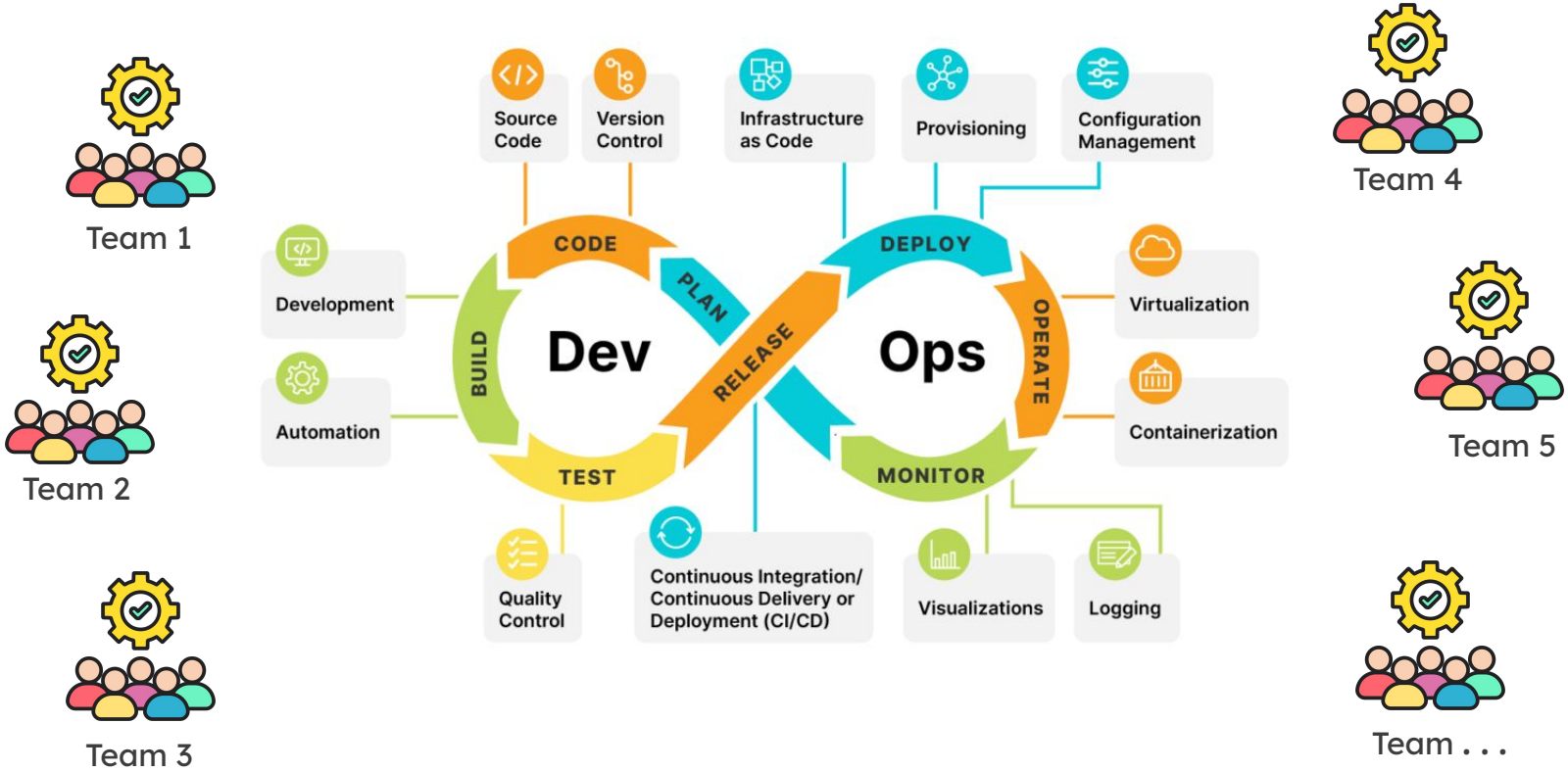
CIKLUM

Business
Data
Application
Technology

Antipattern: Operational Over-Tooling

Antipattern: Operational Over-Tooling

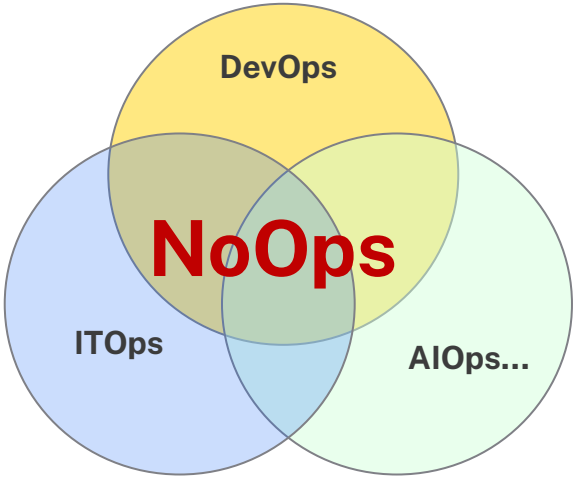
Main problem: Automatisation as much as possible



Antipattern: Operational Over-Tooling

Main problem: Trying to implement NoOps

Fear
Of
Manual
Intervention



Benefits:
Maximized Development Time
No Manual intervention
Full Cloud Capacity

Challenges:
Increased Workload
Decreased Security (potentially)
Lack of Compatibility

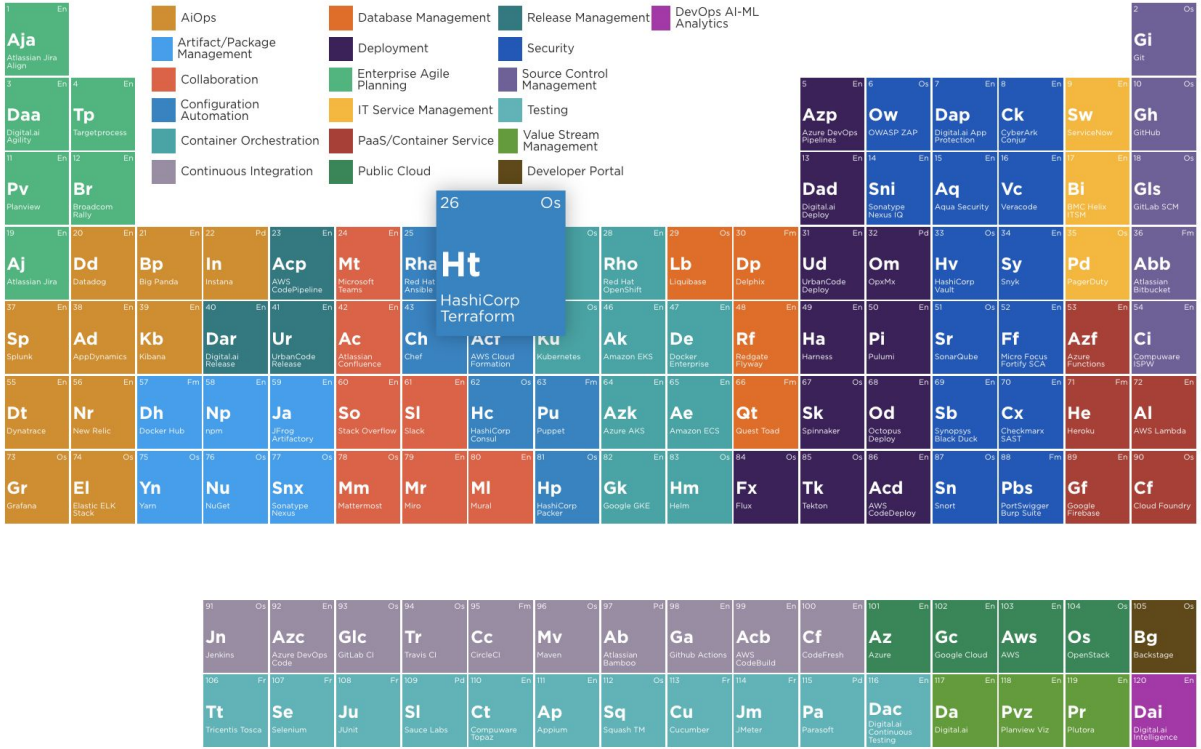
Pitfall
"NoOps Mirage"

Underestimation of the complexities involved in fully automating operations, that's why we have:

- again manual interventions
- maintainability cost increase
- Incidents increase

Antipattern: Over-Tooling Overload

So engineers start searching more tools…



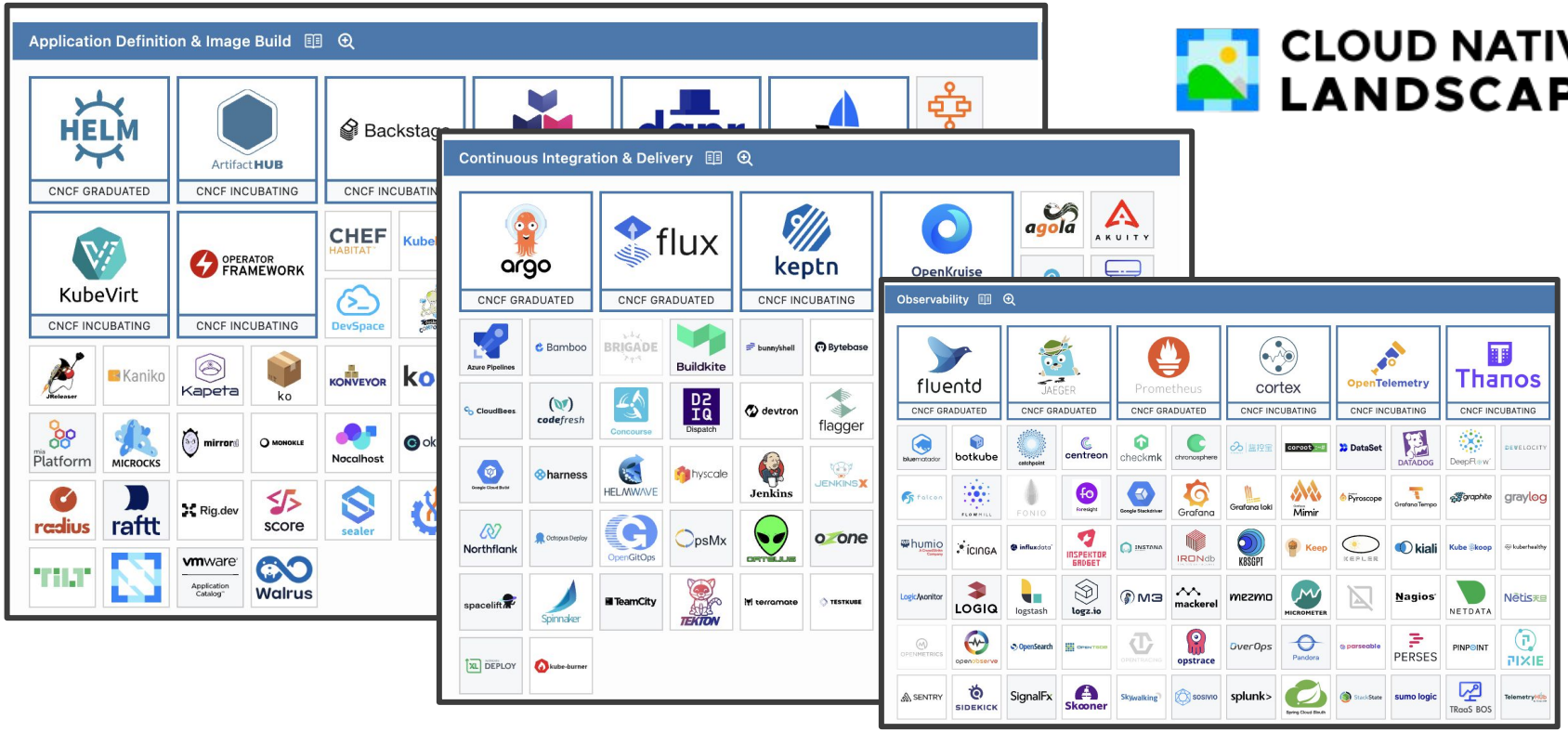
https://digital.ai/learn/devsecops-periodic-table/

Antipattern: Over-Tooling Overload

... and more ....

CIKLUM

CLOUD NATIVE LANDSCAPE



https://landscape.cncf.io

Antipattern: Operational Over-Tooling

And implementation more tools can generate more problems (pitfalls and antipatterns)

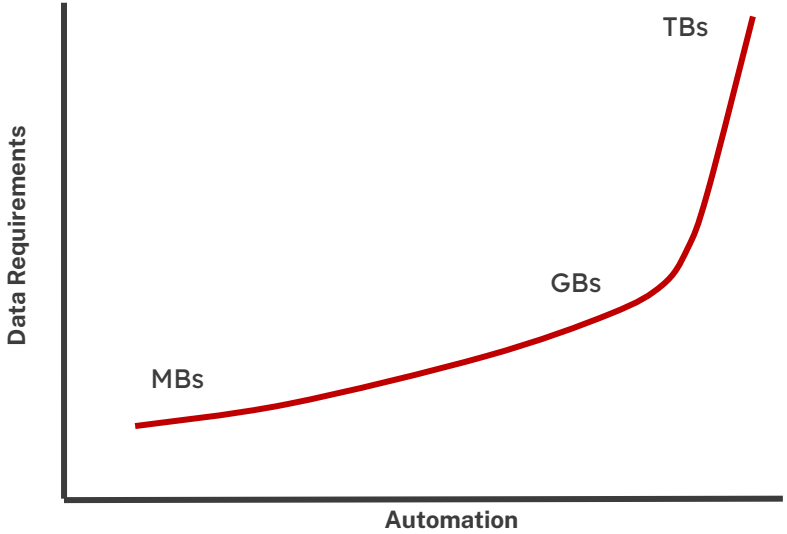
Antipattern 1: “Focus on Tools but not people”

DevOps culture implementation focus on tooling without addressing the importance of having their teams be in the flow and happy.

Antipattern 2: “DevOps is only automation”

You need to step back at first, and make sure you understand all the processes inside the team/company, so you are sure on what you will be automating and what challenges you will be dealing with.

Pitfall 3: “Data Explosion through automation”



Pipelines/builds, Artefacts, Log files, Customer information, Geolocation data, Raw survey data, Financial statements, Emails, Old documents/notes and other files

Antipattern: Operational Over-Tooling

Measurable Metrics to support in identification of anti-pattern

Tool Utilization

Obsolete Tool Count,

Tool Usage Frequency,

% of a tool's features that are actively used,

% of team members actively using each tool

Integration and Maintenance

Integration Time and Effort,

Tool Downtime/Failure Rate,

Time spent managing dependencies and ensuring tool compatibility,

Tool Update Frequency and Update Time,

User Satisfaction and Usability Score,

Support Ticket Volume - # of internal support tickets or help requests related to tool usage or issues

Cost and Resource

Tool Licensing Costs,

Training and Support Costs,

Resource Utilization - cost and % of system resources consumed by tool agents, services, or background processes

Antipattern: Operational Over-Tooling

How to mitigate or fix this

CIKLUM

1. Conduct a **Tools Audit** and **Document (SBOM)** each tools purpose, usage frequency, value provided to team.
2. Implement **Governance for Tool Management**:
   - Introduce a governance policy to periodically review the toolset and ensure it aligns with current team needs.
   - Create a **DevOps Tooling Committee** responsible for approving new tools, reviewing existing ones, and managing integrations.
3. Define a **Tool Adoption Strategy** (clear criteria for adopting new tools).
4. Consolidate Tools (**standardize** to single or minimum number of tools for each category).
5. **Automate** Integration and Maintenance Tasks (reduce manual effort of setting up tools).
6. **Continuous Monitor** and **Measure** tools usage and team productivity.
7. Improve **Documentation** and **Knowledge Sharing**.
8. **Train** and **Onboard** new team members effectively.

CIKLUM

What’s next . . .

What do you need to do?

CIKLUM

What are antipatterns and pitfalls?

Antipatterns and Pitfalls catalogues

What and how should I measure to determine if an antipattern is present in my environment?

Metrics and Observability strategy

What should I do, if it exists, to fix the situation?

Actions Plan

CIKLUM

Any questions?

CIKLUM

Share your feedback!

CIKLUM

Join our team

CIKLUM

Thank you!

Oleksandr Savchenko
LinkedIn:
https://www.linkedin.com/in/o-savchenko/