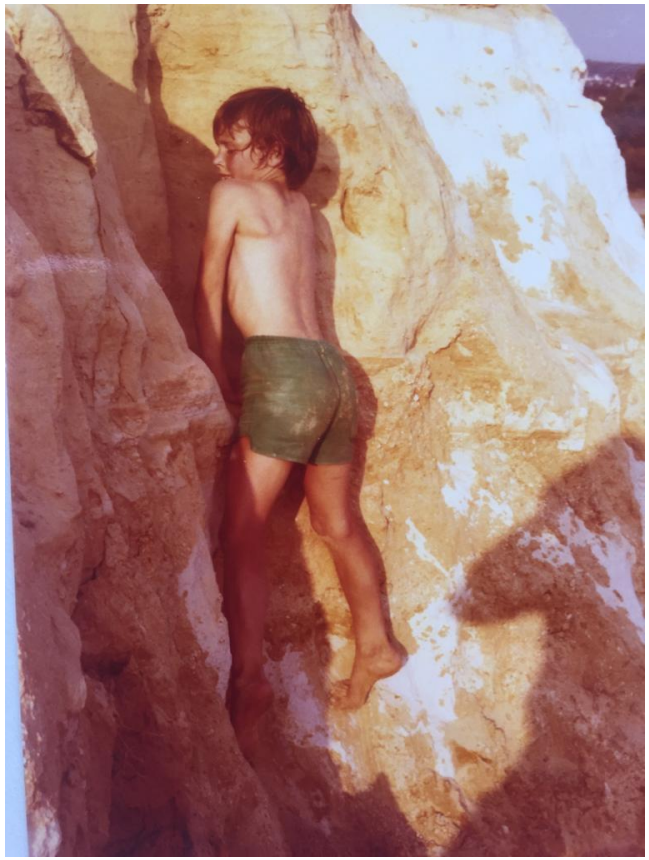
Swiss Association of Computer Science in Sports

Micro-Services based Architectures

Ingo Arnold
Biel, November 26th 2019

Me

*An affinity to **Sports** and an affinity to **Architecture***



Presentation

Purpose & Agenda

Architecture Styles

Overview

"Micro-Services based Architecture" is not a technology-stack, nor is it a product or any other silver bullet promise equipped with a price tag which you can buy from a vendor

«Micro-Services based Architecture» **is an** Architecture Style

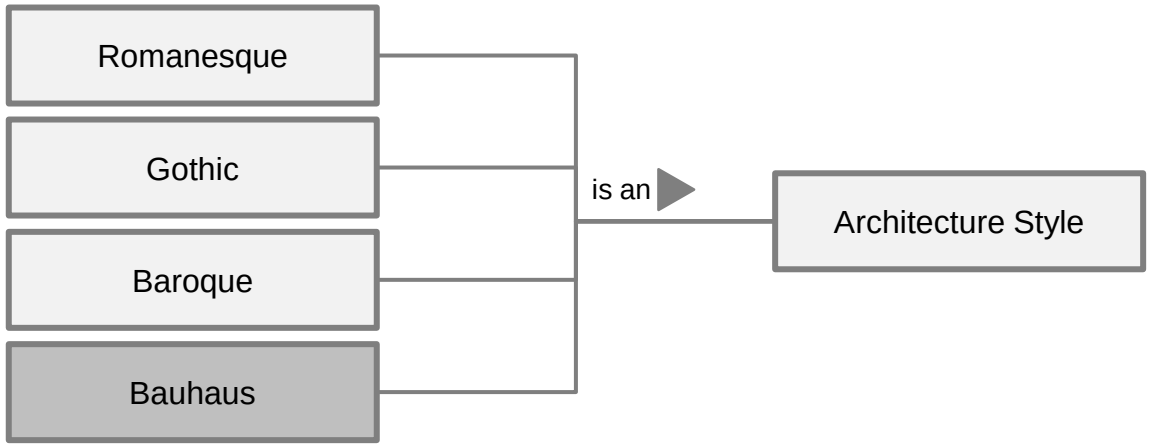
You **cannot buy Architecture Styles** like you can buy IT-Solutions OTS (off-the-shelf)

Architecture Styles

Analogy

Architecture Styles are proven architecture designs, explicit about assumptions, adaption & application, as well as consequences & implications when being adapted

«Bauhaus», for example, **is an** Architecture Style in the traditional architecture domain



Architecture Styles

Analogy

Architecture Styles are proven architecture designs, explicit about assumptions, adaption & application, as well as consequences & implications when being adapted

«Bauhaus», for example, **is an** Architecture Style in the traditional architecture domain

Architecture Style «Bauhaus» (principle: «form follows function»)

Bauhaus

Originally an art school in Germany in the early 1900s the Bauhaus movement held the idea that all art and technology would be unified under the idea of simplistic design and mass-production.

Rejecting decorative details the designs favored function. Flat roofs and cubic shapes were key. The Bauhaus principles of cubic shapes and angles can be seen in the modernist designs

Key feature: Cubic shapes, primary colors of red, blue and yellow, open floor plans, flat roofs, steel frames, glass curtain walls.

instance of

Architecture Styles

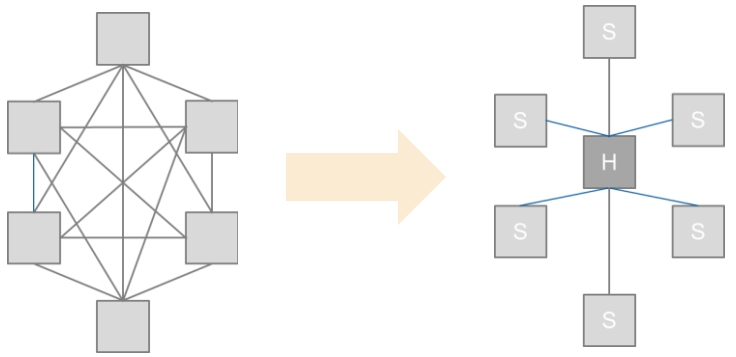
In IT

Architecture Styles and Architecture Patterns have been adapted in IT as a technique in the early 1990ies. Architecture Styles are perfect means to study architecture designs as well as to guide their informed & conscious elaboration

Architecture Style(s) Catalogue

- Publish Subscribe
- N-Tier
- Hub and Spoke
- Layered System
- Peer-2-Peer
- Blockchain
- Micro-Services

«Hub and Spoke» Architecture Style



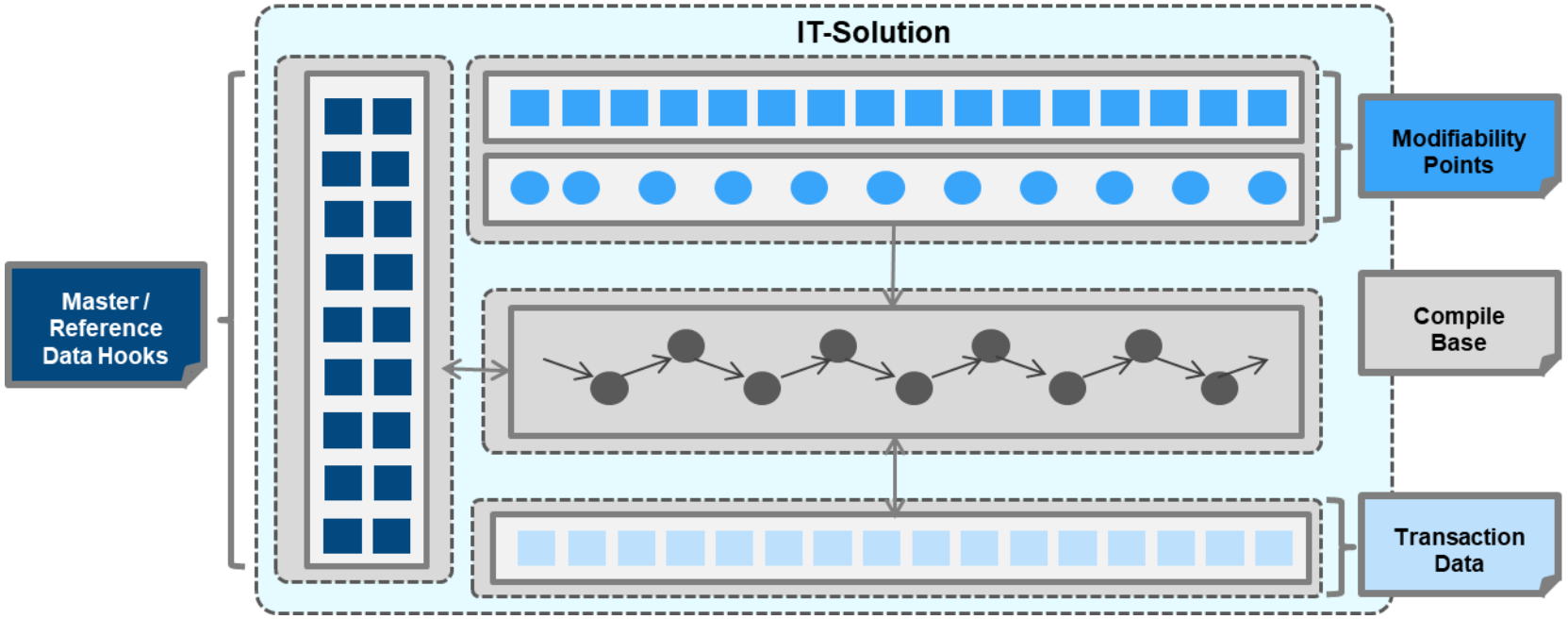
- Need: decrease complexity by decreasing number of interconnections between related systems
- Architectural Approach:
  - Introduce turntable component (Hub)
  - Establish canonical info-exchange norm
  - Connect each system (Spoke) with the turntable (Hub)

IT-Solutions in General

IT-Solutions – generic view (building block types)

IT-Solutions are constituted of a generic foundation (**Compile Base** ( ), **Modifiability Points** ( ) they foresee to be tailored, **Transaction Data** ( ) they produce and **Master / Reference Data Hooks** ( ) to be integrated into a broader context and ecosystem

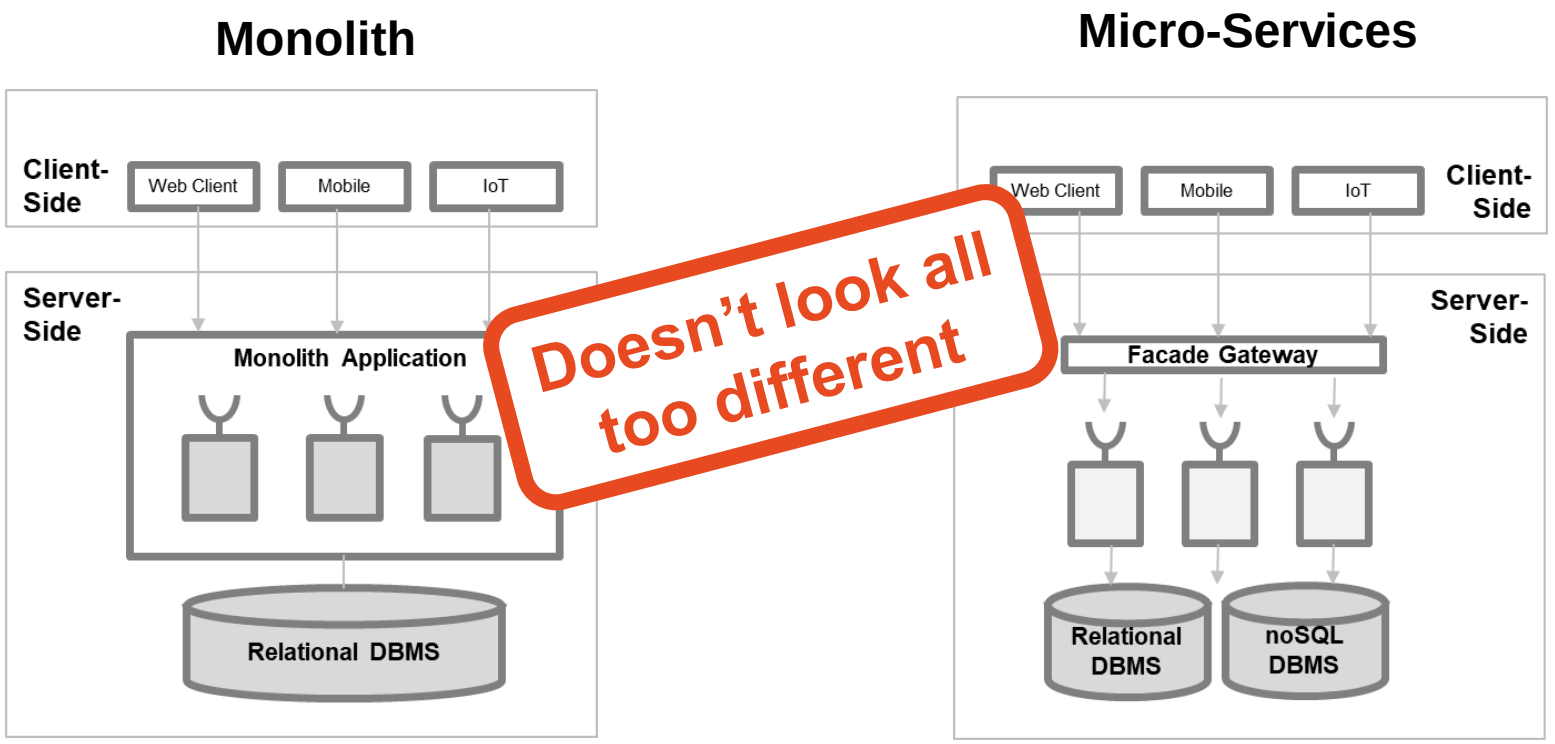
All IT-Solutions share this generic foundation – irrespective of Architecture Style(s) they adapt



Micro-Services Architecture Style

Components View

Looking at depictions of the Micro-Services Architecture Style (the one below is from IBM Developer Works) differences between Micro-Services architecture design and other architecture design approaches seem rather marginal

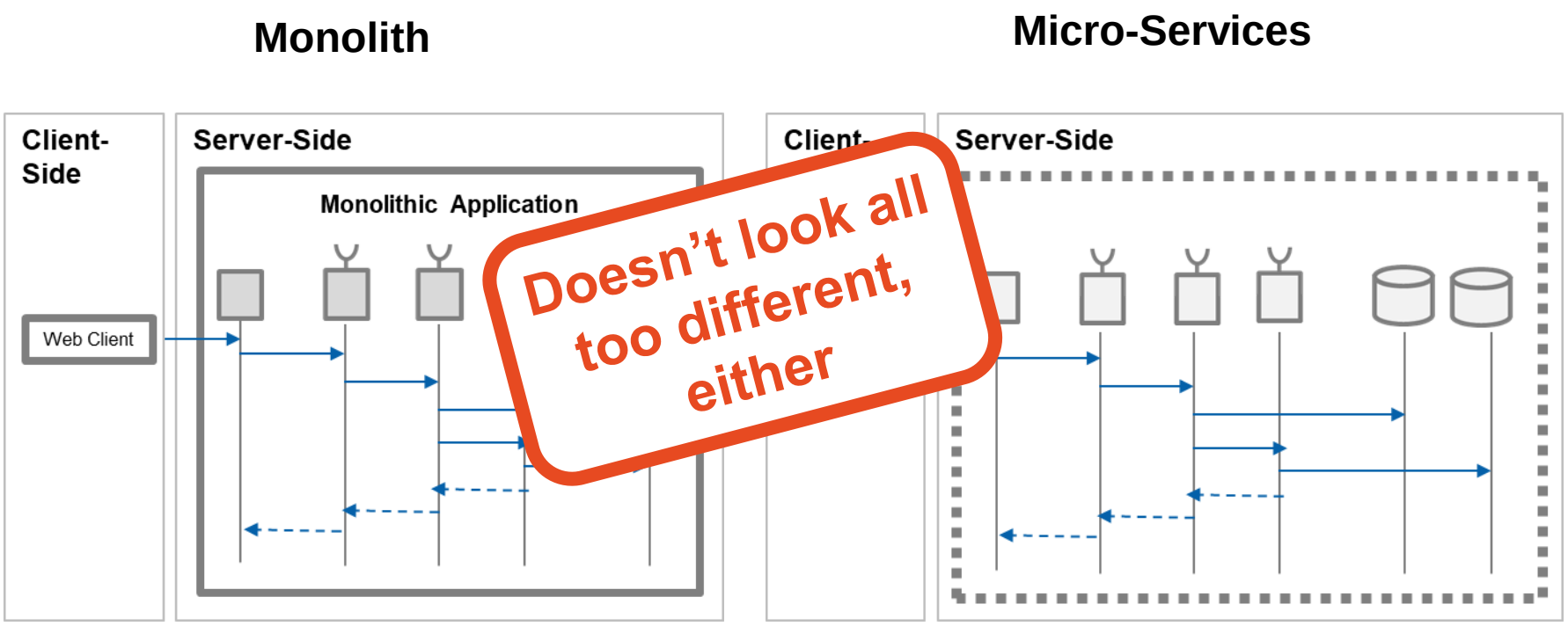


IBM Developer Works

Micro-Services Architecture Style

Components Cooperation View

Looking at depictions of the Micro-Services Architecture Style (the one below is from IBM Developer Works) differences between Micro-Services architecture design and other architecture design approaches seem rather marginal

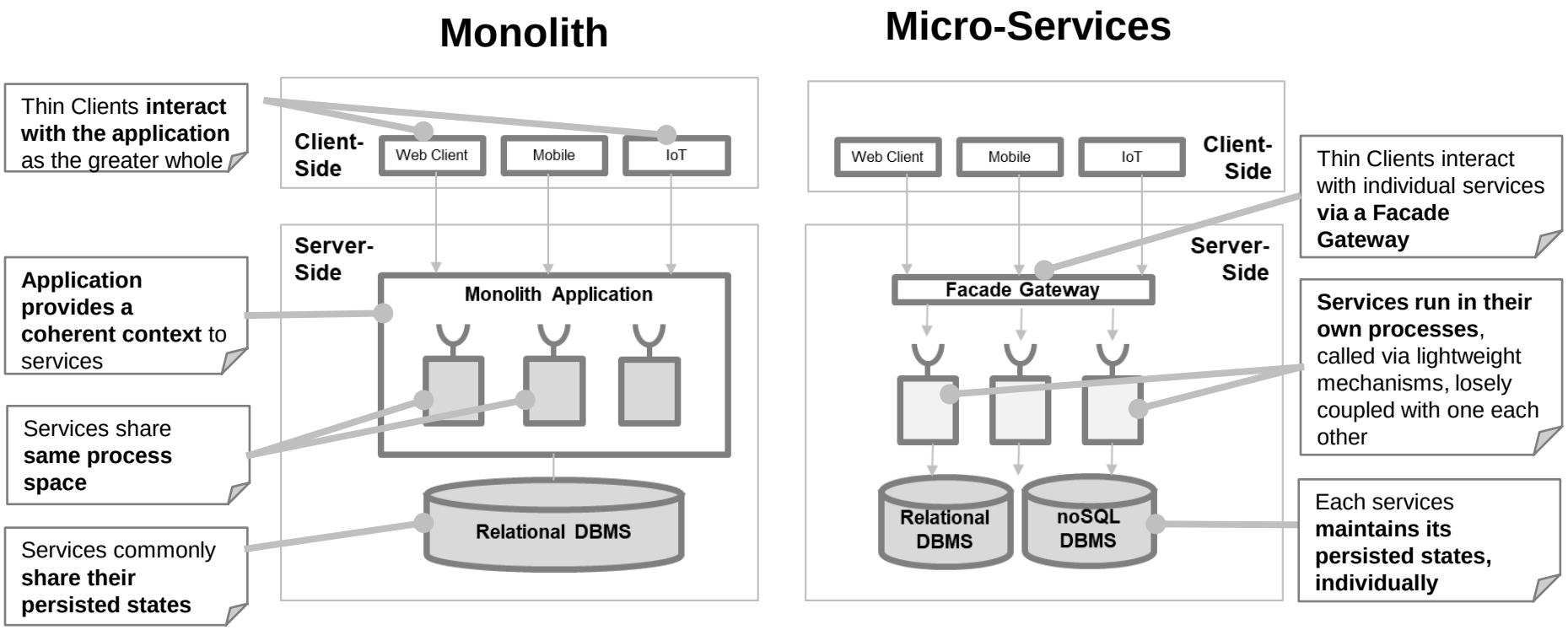


IBM Developer Works

Micro-Services Architecture Style

Overview

Well, of course there are differences. However, they remain invisible in the taken syntactic perspective and only unveiled when we zoom deeper into the component semantics

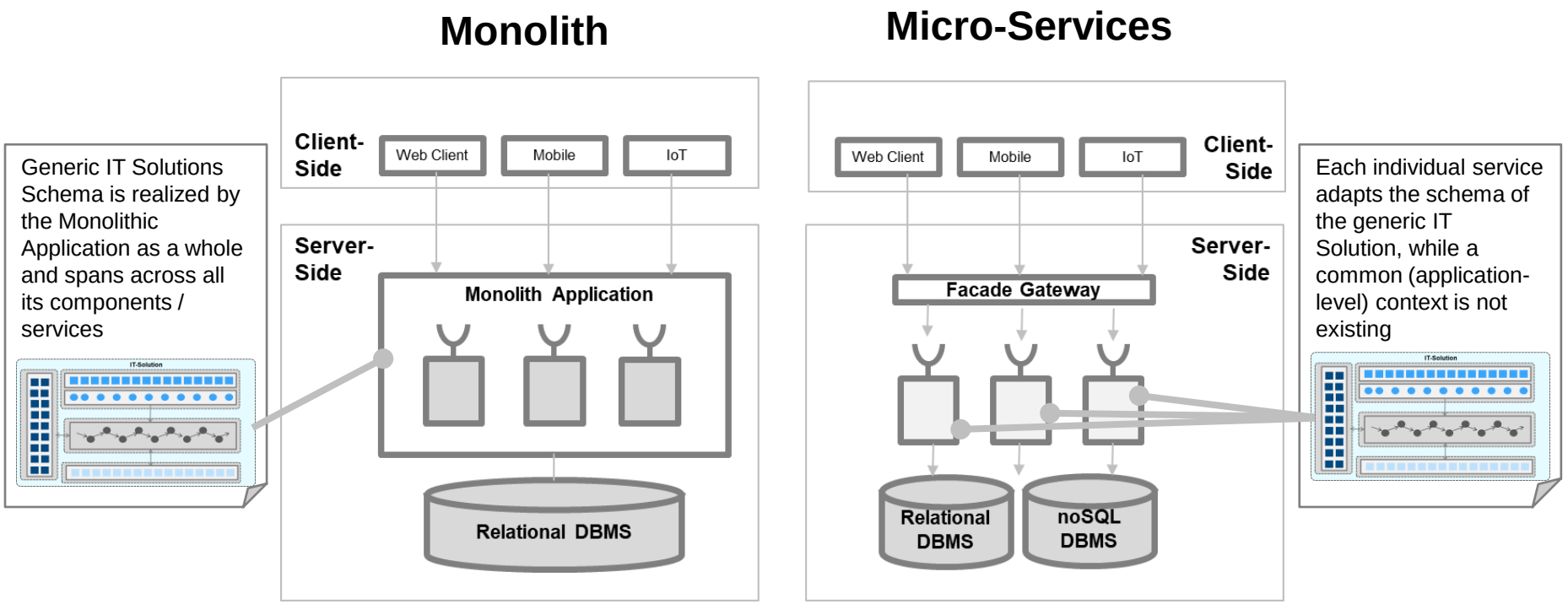


IBM Developer Works

Micro-Services Architecture Style

Overview

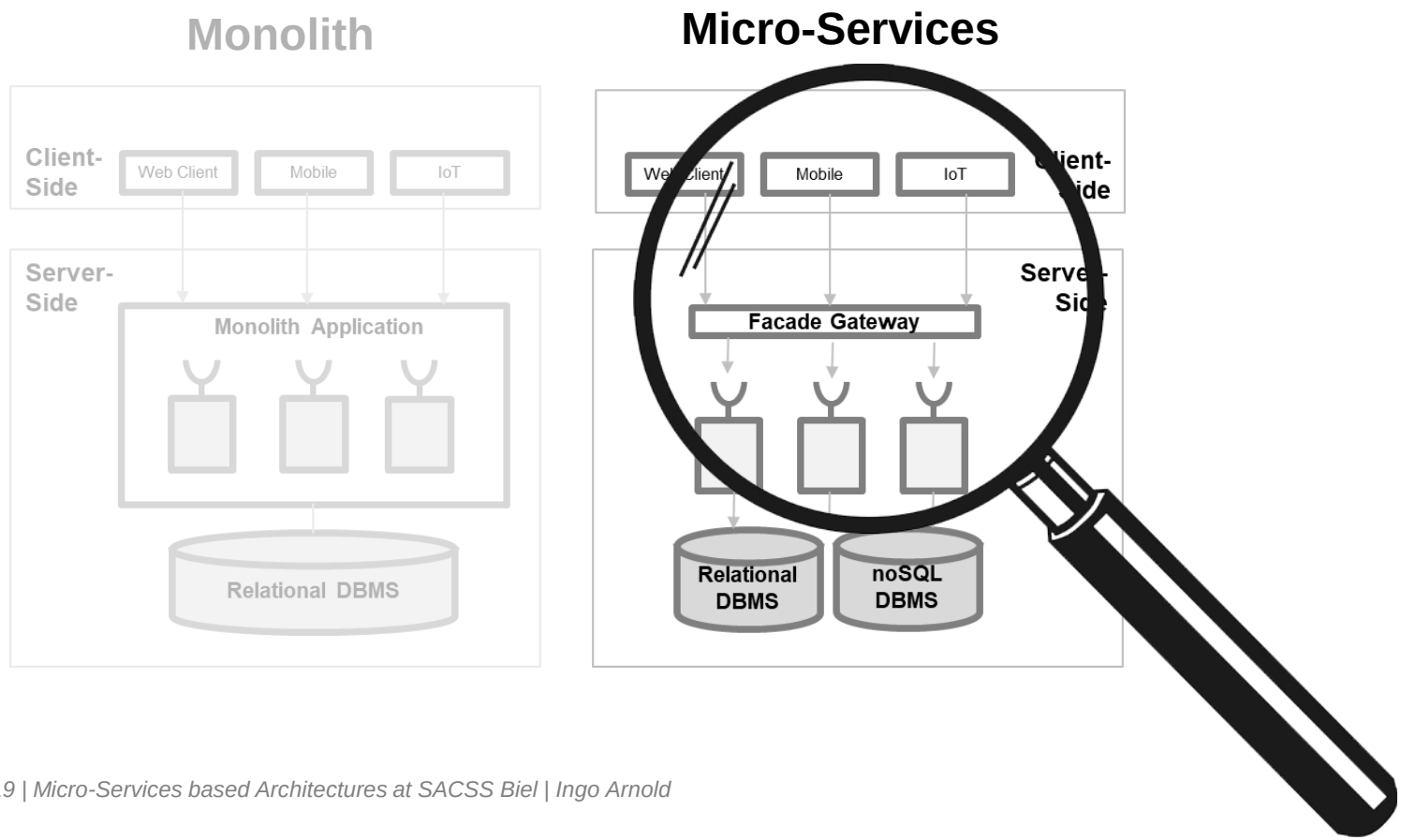
Well, of course there are differences. However, they remain invisible in the taken syntactic perspective and only unveiled when we zoom deeper into the component semantics



Micro-Services Architecture Style

Overview

While there is no precise definition of this architectural style, there are common characteristics like componentization, business capability centricity, approaches to developing services, etc. sufficiently defining the Micro-Services Architecture Style

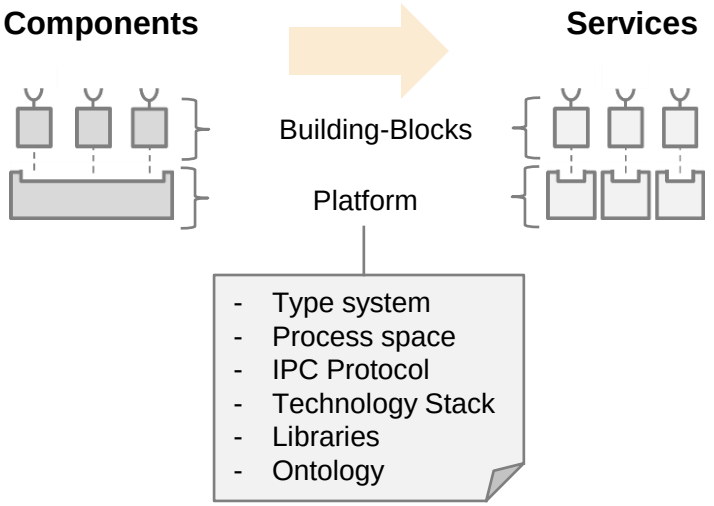


Micro-Services Architecture Style

Overview

Componentization via Services gives you independently deployable services with more explicit interfaces (→ loser coupling). However, the increased flexibility and scalability come at the price of more costly IPC interactions across services

- Componentization via Services
- Organized around Business Capabilities
- Products not Projects
- Smart endpoints and dumb pipes
- Decentralized Governance & Ontology
- Infrastructure Automation
- Design for failure
- Sharing and discovering Services



Challenges
- Confirm you really need utmost flexibility & scalability in your design
- Optimize your interfaces towards lesser calls
- Ensure your development teams are skilled top-2-toe in all required technology fields and across BUILD and RUN

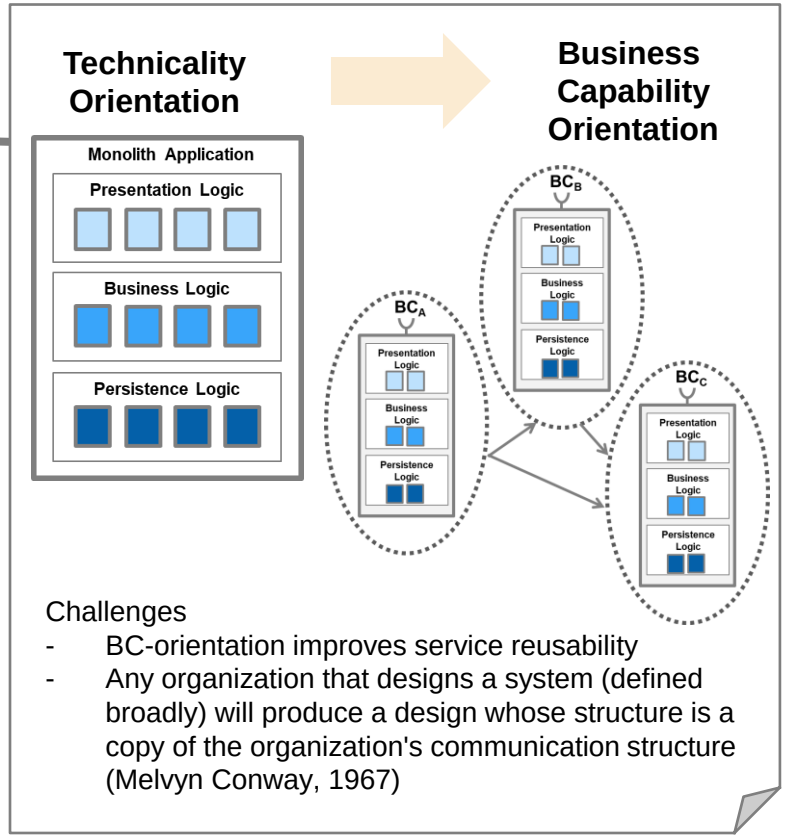
Martin Fowler (Micro-Services, 2014)

Micro-Services Architecture Style

Overview

Mutually aligning between organization and business capability design across your entire domain enables you to harvest service reusability / business adequacy

- Componentization via Services
- Organized around Business Capabilities
- Products not Projects
- Smart endpoints and dumb pipes
- Decentralized Governance & Ontology
- Infrastructure Automation
- Design for failure
- Sharing and discovering Services



Challenges

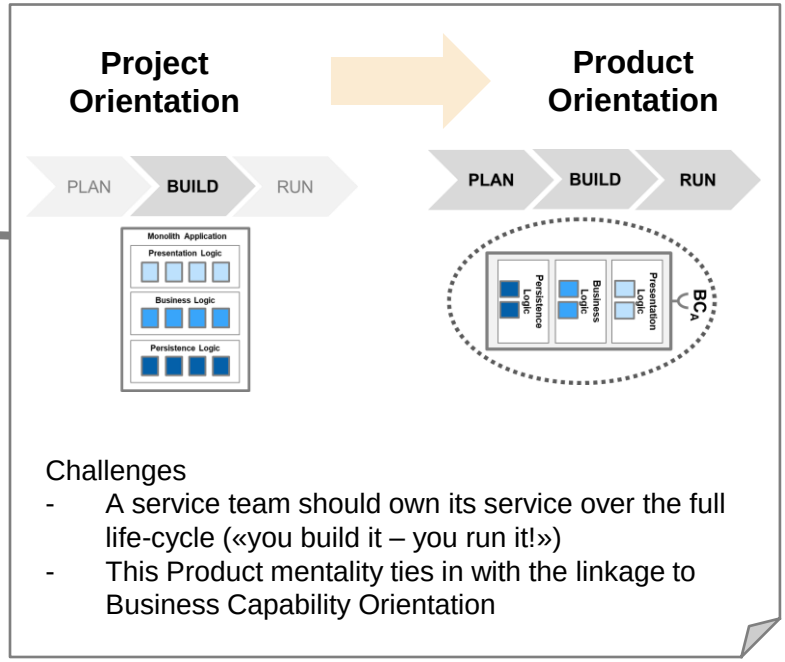
- BC-orientation improves service reusability
- Any organization that designs a system (defined broadly) will produce a design whose structure is a copy of the organization's communication structure (Melvyn Conway, 1967)

Micro-Services Architecture Style

Overview

Ensure every service team owns their service(s) over the entire life-cycle adapting Amazon's "you build it – you run it"-principle

- Componentization via Services
- Organized around Business Capabilities
- Products not Projects
- Smart endpoints and dumb pipes
- Decentralized Governance & Ontology
- Infrastructure Automation
- Design for failure
- Sharing and discovering Services



Challenges
- A service team should own its service over the full life-cycle («you build it – you run it!»)
- This Product mentality ties in with the linkage to Business Capability Orientation

Micro-Services Architecture Style

Overview

Make services as smart and fully accountable for the domain logic they represent as ever possible. Ensure service-orthogonality adapting Unix Shell's Pipes & Filters metapher

- Componentization via Services
- Organized around Business Capabilities
- Products not Projects
- Smart endpoints and dumb pipes
- Decentralized Governance & Ontology
- Infrastructure Automation
- Design for failure
- Sharing and discovering Services

Dumb endpoints smart pipes → **Smart endpoints dump pipes**

Challenges

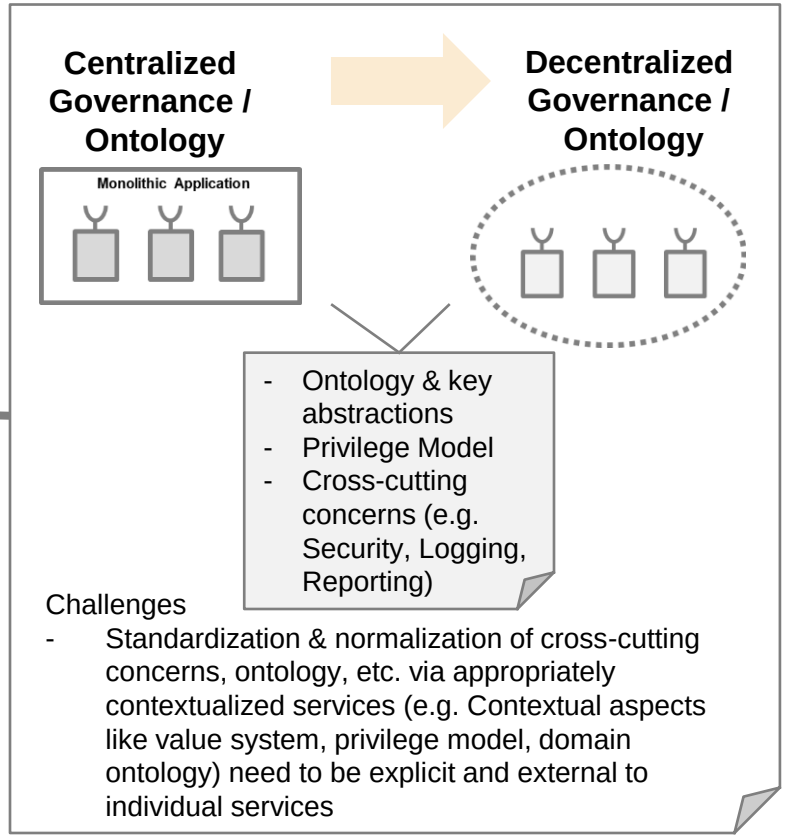
- Integration approaches like ESBs favored sophisticated facilities for message routing, choreography, transformation, and applying business rules
- Services aim to be as decoupled and as cohesive as possible
- Services own their own domain logic and act as filters in the classical Unix sense - receiving a request, applying logic as appropriate and producing a response
- Beware the problems of chattiness though when changing a monolithic application 1:1 into a service-oriented solution

Micro-Services Architecture Style

Overview

Lacking the concept of application-wide context leads to the need for contextualizing services with regards to domain ontologies, privilege models, value systems, etc.

- Componentization via Services
- Organized around Business Capabilities
- Products not Projects
- Smart endpoints and dumb pipes
- Decentralized Governance & Ontology
- Infrastructure Automation
- Design for failure
- Sharing and discovering Services



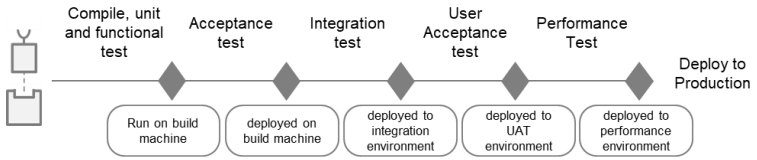
Micro-Services Architecture Style

Overview

High expectations towards the flexibility and scalability of Micro-Services Architecture based designs requires more sophisticated and integrated development, delivery and deployment work-benches

- Componentization via Services
- Organized around Business Capabilities
- Products not Projects
- Smart endpoints and dumb pipes
- Decentralized Governance & Ontology
- Infrastructure Automation
- Design for failure
- Sharing and discovering Services

Continuous Delivery & Integration



Challenges

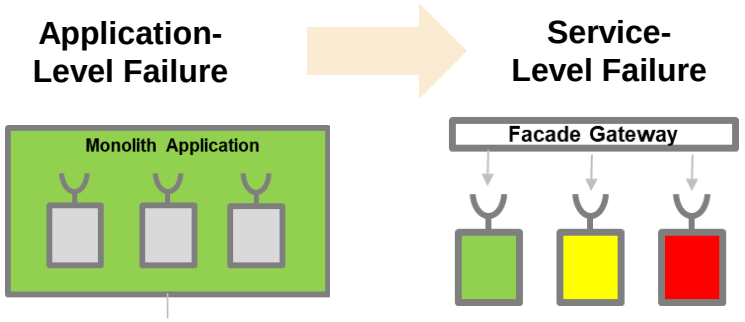
- Evolution of cloud and virtualization technologies has reduced the operational complexity of building, deploying and operating units of functionality (i.e. services)
- Use Continuous Delivery and Continuous Integration as infrastructure automation techniques.
- Pay attention to testing techniques and ensure operational stability by tightly monitoring vitality of services deployed to production

Micro-Services Architecture Style

Overview

IT-Solutions based end-2-end on autonomous services have significantly higher needs towards the operational stability & recoverability of these services – respectively demanding more sophisticated monitoring & recovery solutions underpinning services

- Componentization via Services
- Organized around Business Capabilities
- Products not Projects
- Smart endpoints and dumb pipes
- Decentralized Governance & Ontology
- Infrastructure Automation
- Design for failure
- Sharing and discovering Services



Challenges

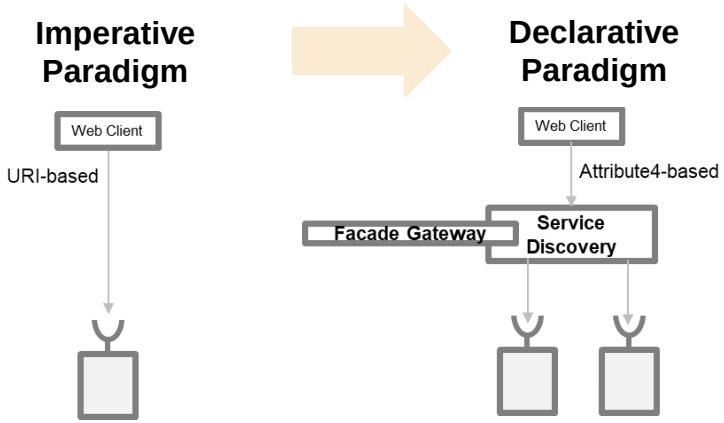
- Design IT-solutions adapting Micro-Services Architecture so that they are resilient to services being down or failing
- Cross-cutting availability techniques can be off-loaded to Facade Gateways (e.g. Redundant service set-up and round-robin equivalent services)
- Ensure adequate testing and monitoring is in place during design- as well as run-time

Micro-Services Architecture Style

Overview

Sharing and discovering services assumes that there is a will to share as well as a will to reuse. Service discovery introduces location transparency which is further enhanced by promoting a declarative discovery paradigm (e.g. service-version, -cost, SLA attrib.)

- Componentization via Services
- Organized around Business Capabilities
- Products not Projects
- Smart endpoints and dumb pipes
- Decentralized Governance & Ontology
- Infrastructure Automation
- Design for failure
- Sharing and discovering Services



Challenges

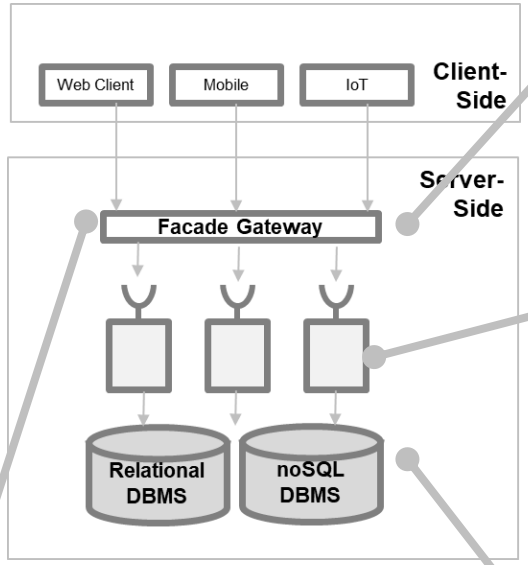
- What motivates someone to advertise and share a service?
- Declarative versus imperative paradigm (decouples requester from provider and enables location-transparency → flexibility, adaptability)
- Consider specific attributes for specific optimization targets and needs (e.g. Service-version, service cost, service-level agreements)

Wrap-Up

Putting it all together

While there is no simple recipe there are few check-points you want to tick-off your Micro-Services Architecture list nevertheless in putting it all together

1. **Lay foundation for basic & common understanding of your domain** – establish a Domain Ontology covering your domain overall in terms of Business Capabilities, inherent value system, privilege model, etc.
2. **Consider organizational set-up consciously** – orientate roles & responsibilities of the overall social system (e.g. Organization, teams) along the BC break-down of your Domain Ontology
3. **Incentive the «social system» to propagate, share, discover and reuse shared services**
4. **Ensure the «social system» is ready for the shift towards service-orientation** and takes full accountabiltiy for their services' life-cycle
5. **Design and operate your «whole»** so
   - It is adaptive to failures or evolution of individual services
   - Cross-cutting ontology and concerns are contextualized
   - Evolution is built in by a respective versioning schema



6. **Enable the discovery of services based on a «declarative discovery paradigm»** adapting domain ontology categories
7. **Design your services** so that ..
   - they realize BCs coherently
   - IPC chattiness and performance are kept to a minimum
   - They cover domain logic autonomously and don't off-load it to inter-connecting pipes
8. **Make operational platform services available across all BC-services** efficiently enabling ..
   - Individual life-cycle management, continuous development, testing & deployment
   - Operational stability
   - Monitoring & Recovery

Wrap-Up

It remains you who owns your IT-Solution's architecture design

An Architecture Style equips you with **potential** benefits of an appropriate design – in order to **actually** harvest an Architecture style's promised highlights **you** (still) need to elaborate the concrete adequate design for your domain

Whether – applying an architecture style – you are creating a solid house or a barrack and shanty remains your accountability

Architecture Style «Bauhaus»
(principle: «form follows function»)

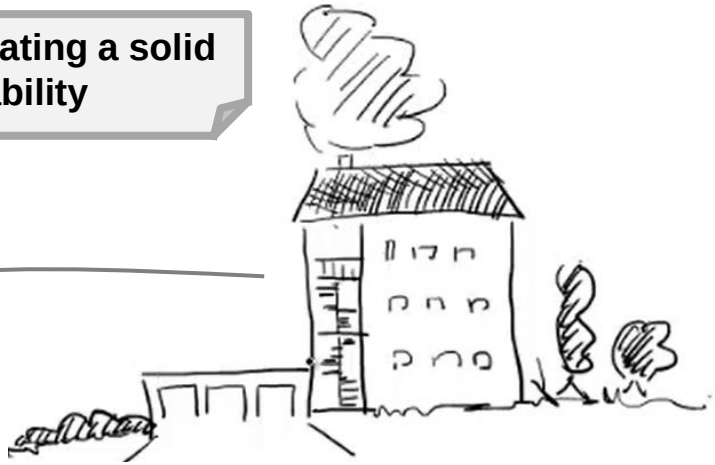
Bauhaus

Originally an art school in Germany in the early 1900s the Bauhaus movement held the idea that all art and technology would be unified under the idea of simplistic design and mass-production.

Rejecting decorative details the designs favored function. Flat roofs and cubic shapes were key. The Bauhaus principles of cubic shapes and angles can be seen in the modernist designs

Key feature: Cubic shapes, primary colors of red, blue and yellow, open floor plans, flat roofs, steel frames, glass curtain walls.

instance of



Wrap-Up

Micro-Services Architecture is no rocket science

Micro-Services based Architecture design is not rocket science – however, if you get your design right, then it's likely you are going to light a rocket

Thank you