Laws, Principles & Practices for Microservices Architecture

Alex Bolboaca
CTO, Trainer & Coach @ Mozaic Works

A Revolution!

We can write in any programming language we like

We work on small things and it will be so much nicer

No testing

We will work faster

Photo by Sam Schooler on Unsplash

At SoCraTes UK 201?

- What are microservices?
- How large are they?
- How do they communicate?
- What issues will we face using them?

Photo by Vadim Bogulov on Unsplash

So, what are microservices?

In short, the microservice architectural style is an approach to developing a single application as a **suite of small services**, each **running in its own process** and communicating with lightweight mechanisms, often an HTTP resource API. These services are **built around business capabilities** and **independently deployable** by fully automated deployment machinery. There is a **bare minimum of centralized management** of these services, which may be written in different programming languages and use different data storage technologies.

-- James Lewis and Martin Fowler (2014)

Modular Architecture

- Strong Boundaries
- Replaceable
- Clear Responsibilities

Module Deployment

- In a namespace with a Facade (a bit forced)
- Library
- OS Service
- Remote service
- [New] Modules as first class citizens in programming languages

Microservices are

Just another iteration of modular distributed architecture,

Taking advantage of advances in automation and cloud services

Back to the fundamentals!

Photo by Paul Kramer on Unsplash

So, a few laws

- The first law of software architecture
- The law of conservation of complexity
- The CAP Theorem
- Conway’s Law
- Low coupling, high cohesion

The first law of software architecture

Everything in Software Architecture is A Trade-Off

Mark Richardson & Neal Ford,
Fundamentals of Software Architecture

Corollary

If an architect thinks they have discovered something that isn’t a trade-off, they haven’t identified the trade-off yet

Mark Richardson & Neal Ford,
Fundamentals of Software Architecture

Second Law of Software Architecture

Why is more important than how

Mark Richardson & Neal Ford,
Fundamentals of Software Architecture

Fred Brooks

- Author of “The Mythical Man-Month”
- Turing Award in 1999
- Died on 17 Nov 2022
- We did a video in his honor
  https://www.youtube.com/watch?v=1XIWZyplgrM

Complexity

Two types of complexity:
* essential – aka the problem complexity
* accidental – aka the solution complexity

Essential complexity is irreducible

Alex's Addendum

Accidental complexity tends to move around the system

Microservices move complexity from development to operations and debugging

Source: https://liberationchiropractic.com/wp-content/uploads/2016/06/Whackamole.jpg

Conway's Law

Any organization that designs a system (defined broadly) will produce a design whose structure is a copy of the organization's communication structure.

Melvin E. Conway, How Do Committees Invent?, 1968

Conway’s Law Applied to Microservices: Fred George

- Low coupling between microservices (aka events)
- Low coupling => low communication between microservices devs
- High parallelization of work
- If microservices are defined

CAP Theorem

It is impossible for a distributed data store to simultaneously provide more than two out of the following three guarantees:

Consistency
Availability
Partition tolerance

Explain CAP

- **Consistency:** ***Every read receives the most recent write or an error***
- **Availability:** ***Every request receives a (non-error) response, without the guarantee that it contains the most recent write***
- **Partition Tolerance:** ***the system continues to operate despite an arbitrary number of messages being dropped (or delayed) by the network between nodes***

CAP Theorem for Microservices

- We need availability
- We need partition tolerance
- => Relax consistency
- => *Eventual consistency*

Photo by Pierre Bamin on Unsplash

Low Coupling, High Cohesion

- Low coupling between microservices
- High cohesion inside a microservice
- For performance reasons, we can use higher coupling within a bounded context

Connascence

Connascence is a software quality metric & a taxonomy for different types of coupling.

3 Axes of Connascence

- **Strength. Stronger connascences are harder to discover, or harder to refactor.**
- **Degree. An entity that is connascent with thousands of other entities is likely to be a larger issue than one that is connascent with only a few**
- **Locality. Connascent elements that are close together in a codebase are better than ones that are far apart.**

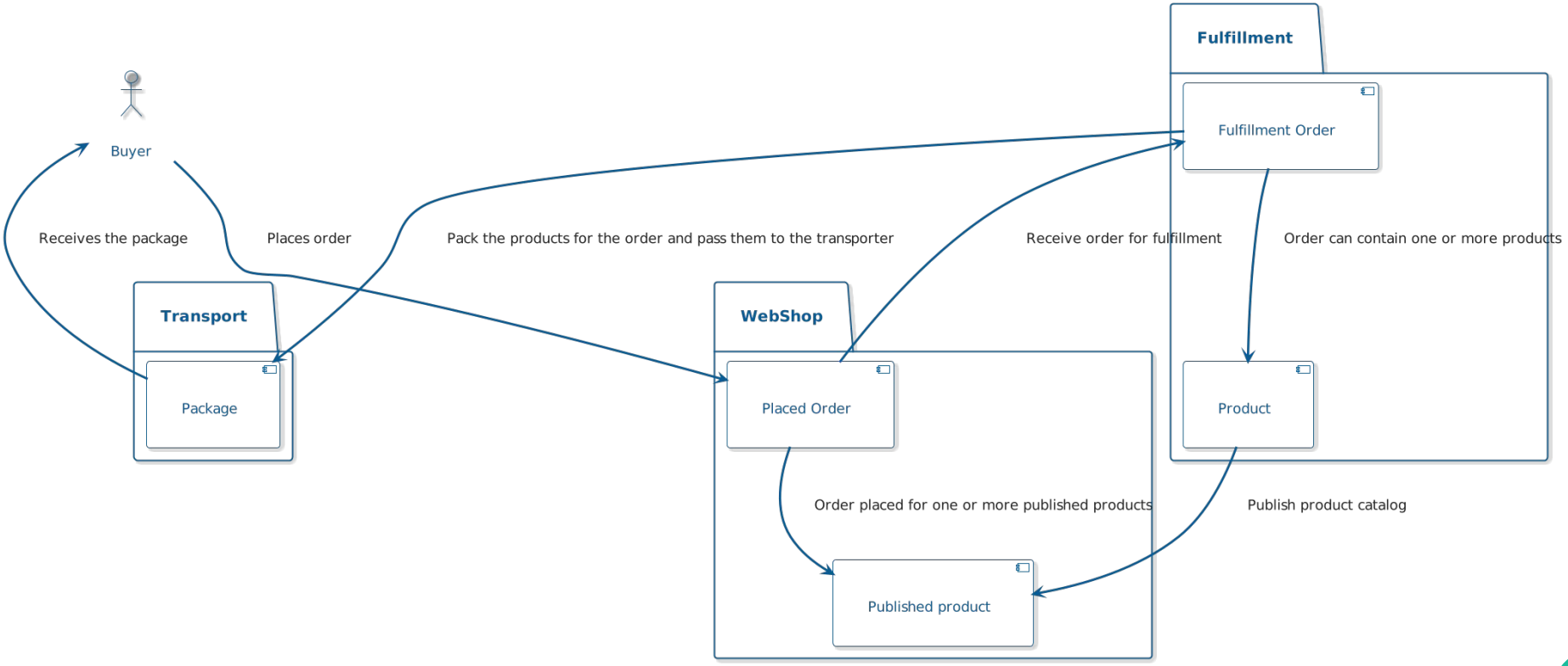
High coupling / connascence => Dependency Hell

Source: turbosquid.com

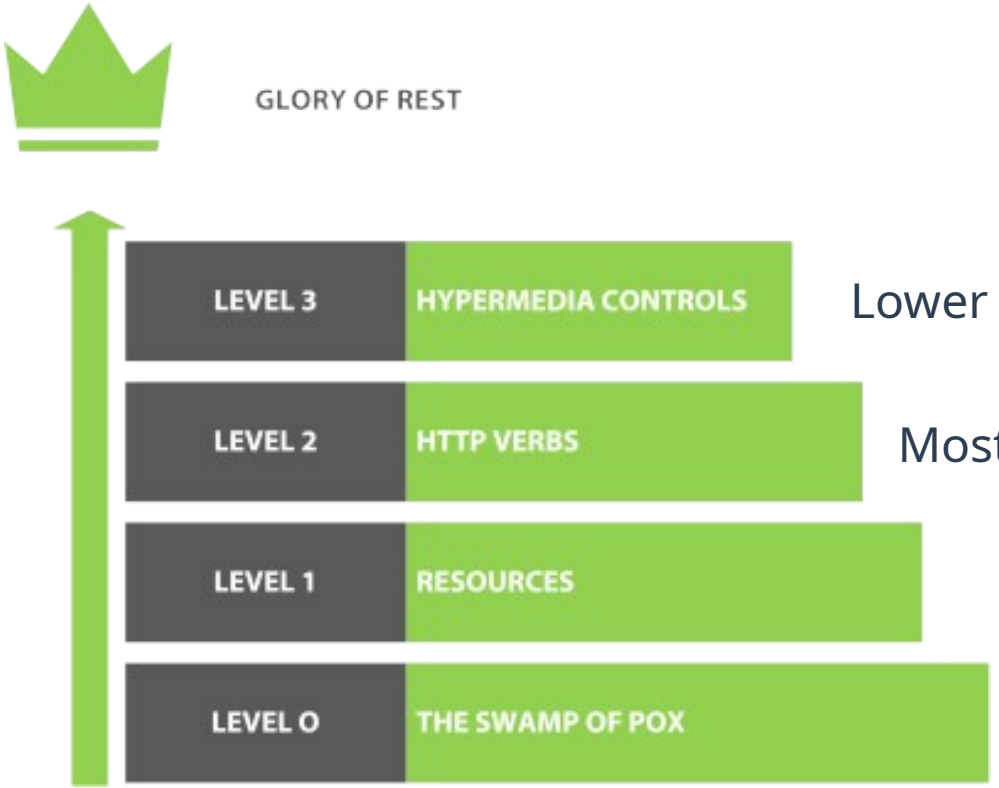
How to architect microservices?

- Domain modeling to identify bounded contexts and domain entities
- Build services around behavior
- Low coupling between bounded contexts
- If performance is needed, slightly higher coupling within a bounded context

Simple Example of Domain Modeling



Levels of REST APIs



Lower coupling

Most organizations are here

Source: Restcase.com

Microservices and OOP

- OOP is not class-oriented programming!
- Alan Kay reinterpreting OOP: “every object should have a url”
- Microservices combine data (data stores) and behavior (implementation)
- Similar to the SmallTalk environment

What have we learned?

Photo by Ilse Orsel on Unsplash

Folklore

Learning in software development happens through folklore

Photo by Karla Vidal on Unsplash

Short-termism

- **Disregard for what came before**
- **Lack of knowledge about history**

Photo by Rachel Hisko on Unsplash

Dire Need of Simplicity

- We feel the need to remove the accidental complexity
- … but go for utopia

Photo by Samantha Gades on Unsplash

Instead...

- Focus on fundamentals that don’t change
- Look at past iterations of the same problem
- Learn other people’s problems
- Don’t blindly copy large organizations

Photo by Marc-Olivier Jaudoin on Unsplash

Remember to

Think. Design. Work Smart.

https://mozaicworks.com
https://youtube.com/@tdws

Learning Programs

- Architecting Microservices
- From Developer to Architect
- Software Architecture Principles
- Serverless Architecture
- Native Cloud Architecture


Questions?