
devops
HEROES


LABS

Bounded Context is not Enough!

Bounded Context & Microservices: a long story of heroic achievement

AGILERELOADED

alberto.acerbis@intre.it

Sponsor & Org

UNIVERSITÀ DEGLI STUDI DI PARMA

engage LABS

innprojekt

iSolutions

daniela malvisi COMMUNICATION

DRILLING PERGEMINE CONTRACTOR

LIME ware

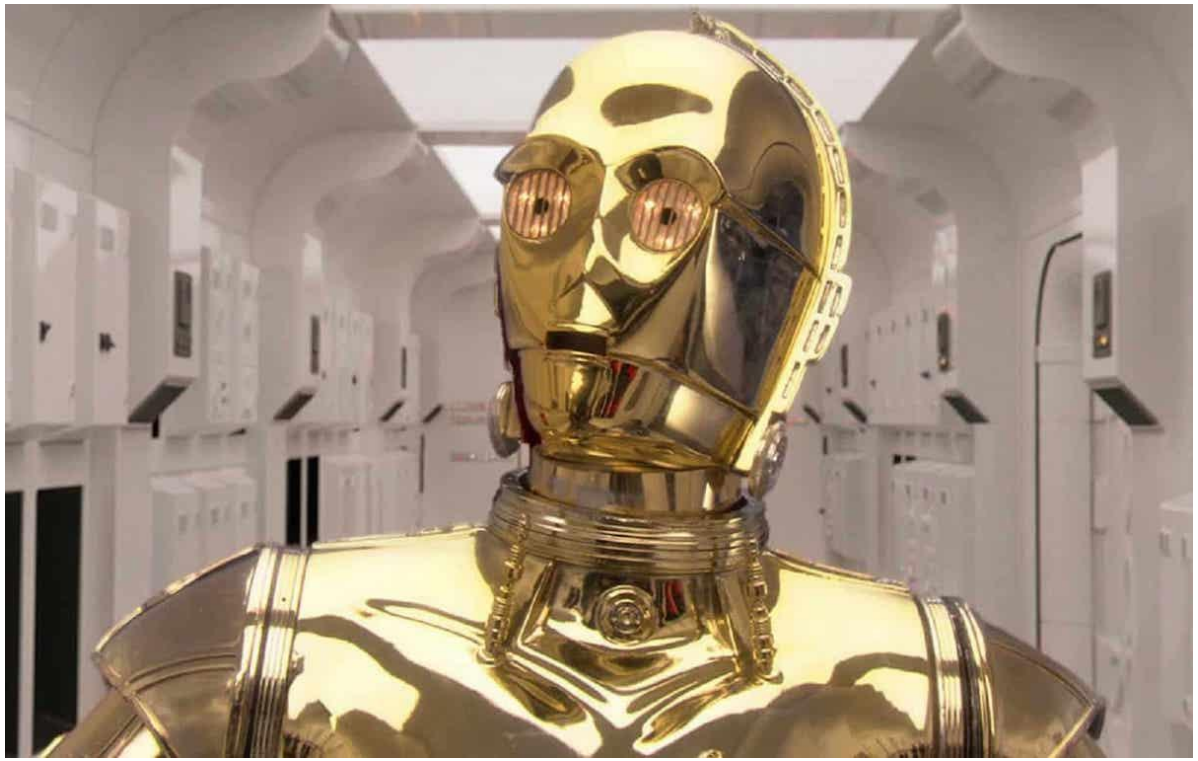
getlatestversion.it

You can sleep …

But the speaker has the right to question you

What do you mean by «Bounded Context»?

It is the focus of DDD's strategic design section which is all about dealing with large models and teams



It is a **boundary** that defines the scope and meaning of a **Ubiquitous Language** for a specific **subdomain or module**

The delimited applicability of a particular model. BOUNDING **CONTEXTS** gives team members a clear and shared understanding of what has to be **consistent** and what can **develop independently**.

What do you mean with «Microservice»?

- Componentization via Services
- Organized around Business Capabilities
- Products not Projects
- Smart endpoints and dump pipes
- Decentralized Data Management
- Infrastructure Automation
- Design for Failure
- Evolutionary Design

Martin Fowler

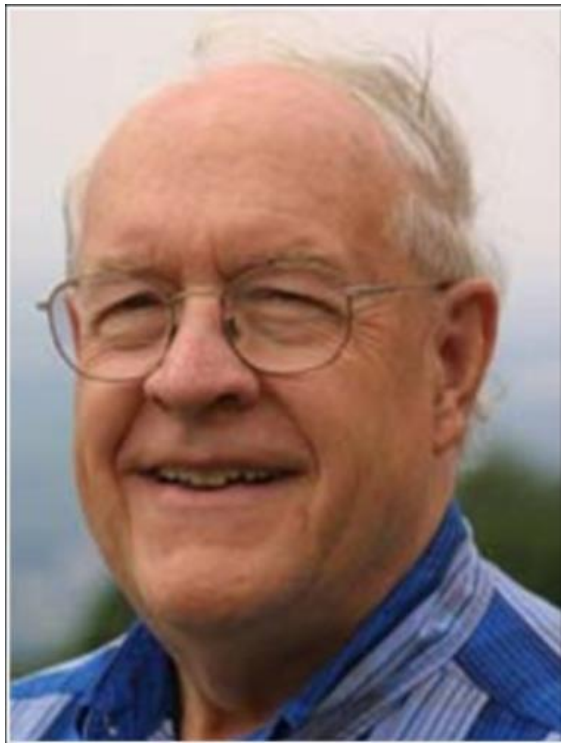
Is there any connection between Bounded Context and Microservices?

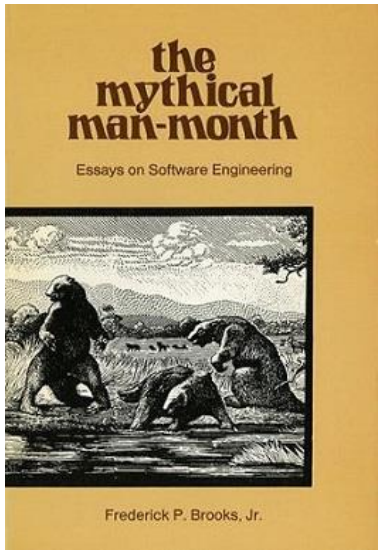
Componentization via Services	No Problem
Organized around Business Capabilities	Match
Products not Projects	No Problem
Smart endpoints and dump pipes	No Problem
Decentralized Data Management	Match
Infrastructure Automation	No Problem
Design for Failure	No Problem
Evolutionary Design	Match

A. Brandolini

Componentization via Services	No Problem
Organized around Business Capabilities	Match
Products not Projects	No Problem
Smart endpoints and dump pipes	No Problem
Decentralized Data Management	Match (?!?!?)
Infrastructure Automation	No Problem (?!?!?)
Design for Failure	No Problem (?!?!?)
Evolutionary Design	Match (?!?!?)

When you fight under distributed systems you need to consider multiple drivers, not just the services that handle specific business functions.





No Silver Bullet

Essence and Accidents of Software Engineering

Frederick P. Brooks, Jr.
University of North Carolina at Chapel Hill

Fashioning complex conceptual constructs is the *essence;* *accidental* tasks arise in representing the constructs in language. Past progress has so reduced the accidental tasks that future progress now depends upon addressing the essence.

Of all the monsters that fill the nightmares of our folklore, none terrify more than werewolves, because they transform unexpectedly from the familiar into horrors. For these, one seeks bullets of silver that can magically lay them to rest.

The familiar software project, at least as seen by the nontechnical manager, has something of this character; it is usually innocent and straightforward, but is capable of becoming a monster of missed schedules, blown budgets, and flawed products. So we hear desperate cries for a silver bullet—something to make software costs drop as rapidly as computer hardware costs do.

But, as we look to the horizon of a decade hence, we see no silver bullet. There is no single development, in either technology or in management technique, that by itself promises even one order-of-magnitude improvement in productivity, in reliability, in simplicity. In this article, I shall try to show why, by examining both the nature of the software problem and the properties of the bullets proposed.

Skepticism is not pessimism, however. Although we see no startling breakthroughs—and indeed, I believe such to be inconsistent with the nature of software—many encouraging innovations are under way. A disciplined, consistent effort to develop, propagate, and exploit these innovations should indeed yield an order-of-magnitude improvement. There is no royal road, but there is a road.

The first step toward the management of disease was replacement of demon theories and humours theories by the germ theory. That very step, the beginning of hope, in itself dashed all hopes of magical solutions. It told workers that progress would be made stepwise, at great effort, and that a persistent, unremitting care would have to be paid to a discipline of cleanliness. So it is with software engineering today.

This article was first published in *Information Processing '86*, ISBN No. 0-444-70077-3, H.-J. Kugler, ed., Elsevier Science Publishers B.V. (North-Holland) © IFIP 1986.

Does it have to be hard?—Essential difficulties

Not only are there no silver bullets now in view, the very nature of software makes it unlikely that there will be any—no inventions that will do for software productivity, reliability, and simplicity what electronics, transistors, and large-scale integration did for computer hardware.

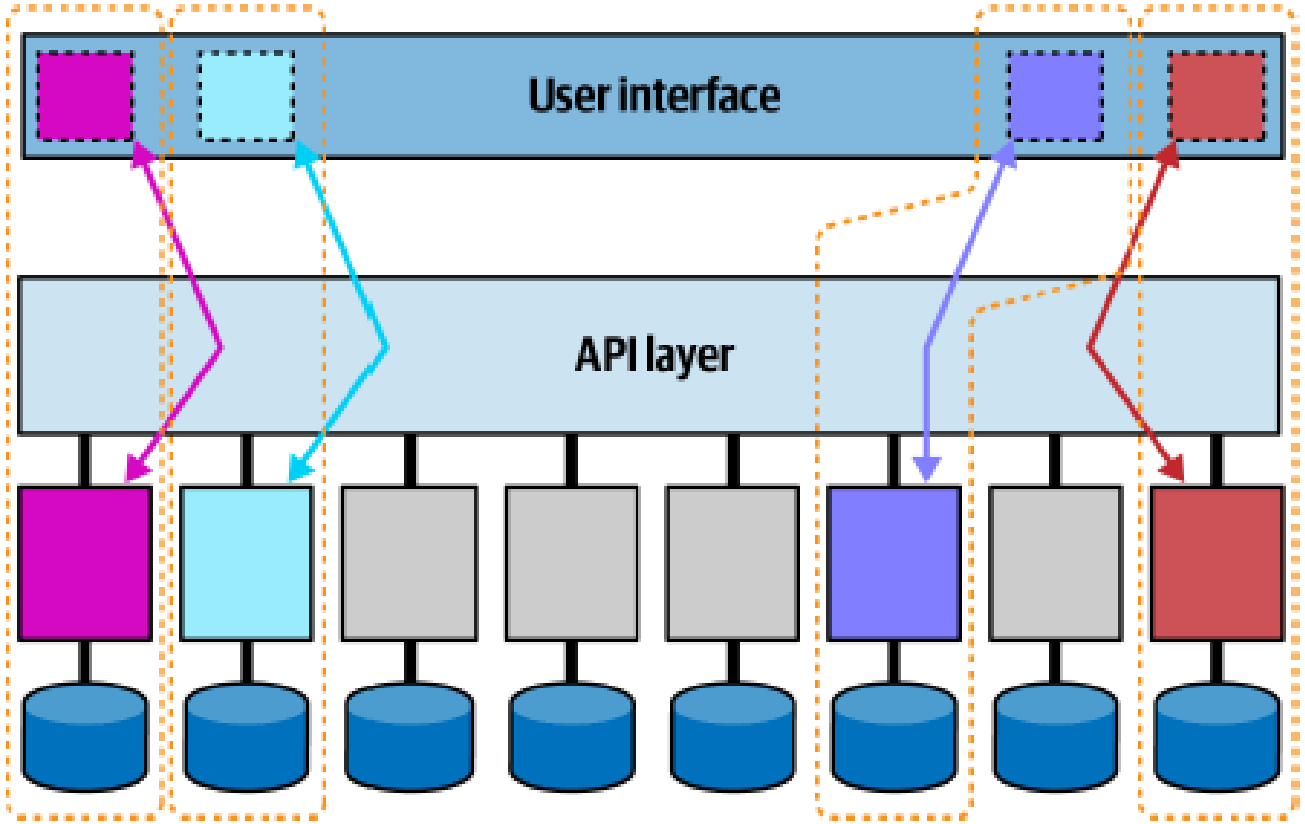
Essence == Business

That is the part that DDD solves

Accident == Operations

Bounded Context is not Enough!

Quantum Architecture



Traps

- Database Refactoring
- Choreography
- Contract Testing
- Continuos Delivery
- Evolvability and Experimentation
- Culture of Experimentation

First Law

Everything in software architecture is a trade-off

Corollary 1

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

Second Law

Why is more important than how

Have you never heard of «Evolutionary Architecture»?

An Evolutionary Architecture supports guided incremental change across multiple dimensions

An Evolutionary Architecture supports **guided incremental** change across **multiple dimensions**

Fitness Functions

An evolutionary computing fitness function characterizes how close a solution is to desire result

An architectural fitness function characterizes how close a system is to the desired architectural characteristics

The code must be maintanable! (What does that mean?)

Outcomes not Implementations

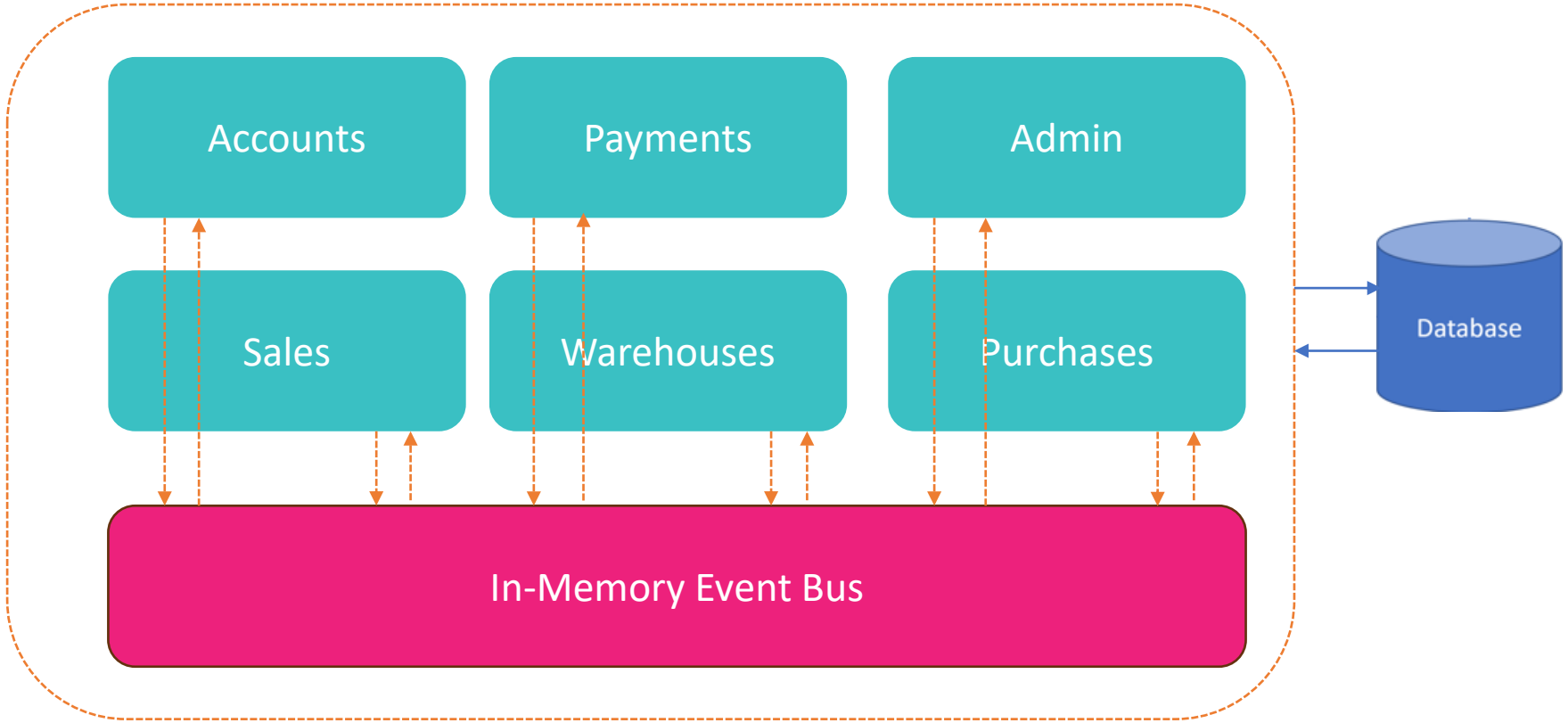
Risk comes from not knowing what you are doing
(Warren Buffet)

Unknow Unknows

Principles

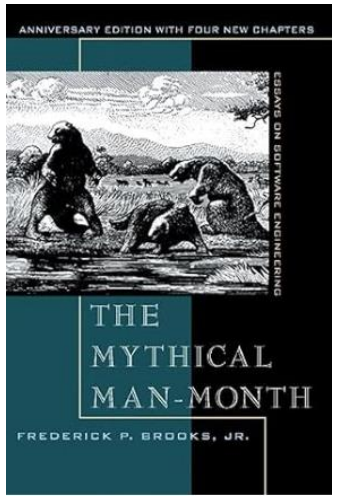
Last Responsible Moment	Architect and develop for evolvability	Postel's Law	Conway's Law
Take decisions at the last responsible moment, because you have the most information	You cannot change a system you don't understand	Be conservative in what you send, be liberal in what you accept	Communication patterns

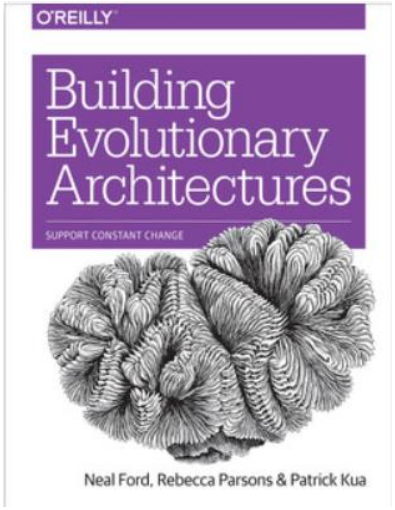
Have you never heard of «Modular Architecture»?

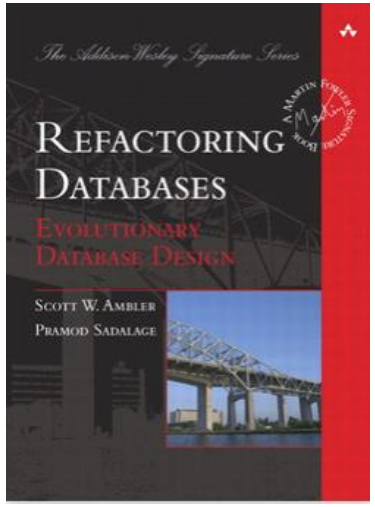

Accounts
Payments
Admin
Sales
Warehouses
Purchases
Database
In-Memory Event Bus

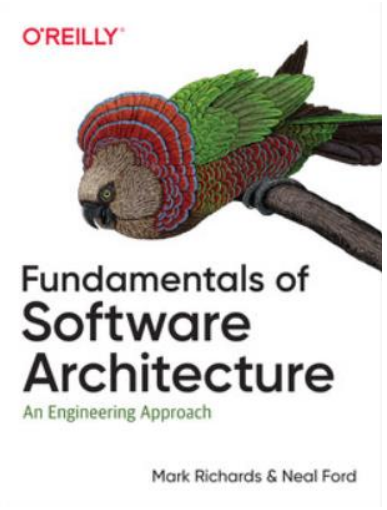
DEMO

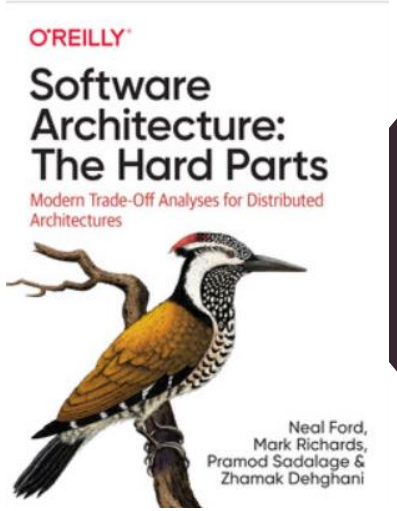
I Sacri Testi











[No Silver Bullet - Essence and Accident in Software Engineering]

[Fitness function-driven development]

[Five Level of Ignorance]

Grazie!!!

IL CORSO

Introduzione a DDD, CQRS ed i loro pattern

Costruire un'applicazione completa (a microservizi) applicando DDD.

Prenota il tuo posto

AGILERELOADED

INTRE CLOUD

alberto.acerbis@intre.it

https://github.com/brewup

https://github.com/cqrs-muflone

https://github.com/ace68

Alberto Acerbis