


The cartoon depicts two characters in a dialogue. The first character asks, "YOU DIDN'T REALLY GET THE REQUIREMENTS. DID YOU?". The second character responds with a large speech bubble containing the text: "BECAUSE I NAMED THESE 550 MESSY LINES OF UNREADABLE CODE doTheseThingsFromTheSpecOnPage317 (int a, double b, Object c) ?". The first character then replies, "YOU NAILED IT". The cartoon is signed "geek & joke" in the bottom right corner of the first panel.



WPS **WORKPLACE**
SOLUTIONS

It's all about the domain, honey !
Experiences from 15 years of Domain-Driven Design

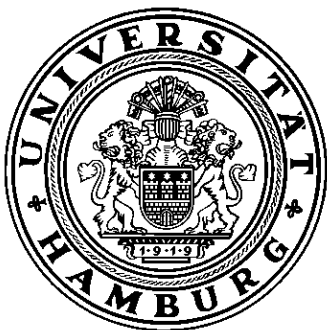
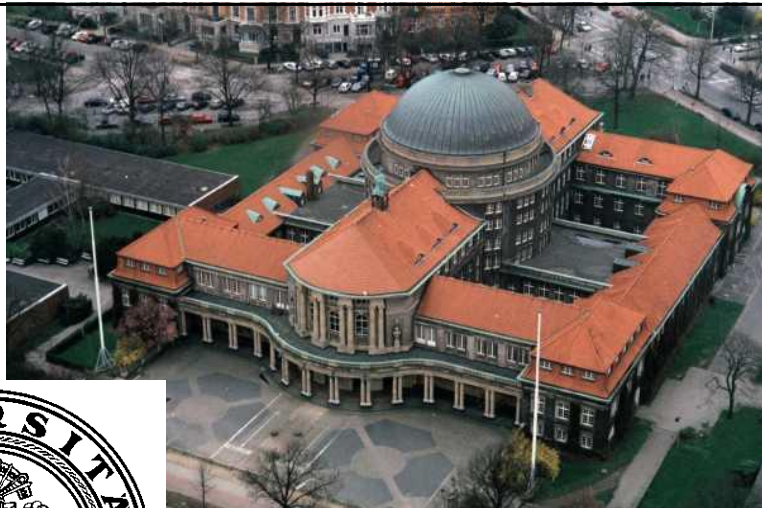
Dr. Carola Lilienthal @cairolali cl@wps.de

WPS - Workplace Solutions GmbH // Hans-Henny-Jahnn-Weg 29 // 22085 HAMBURG

@cairolali



Quelle: <http://www.schulbilder.org/malvorlage-informatiker-i10418.html>







15

»Enjoyable
Business Software«

16



17

SUSTAINABLE ARCHITECTURE



Long useful life → Investment pays off



Lowest possible maintenance and extension costs

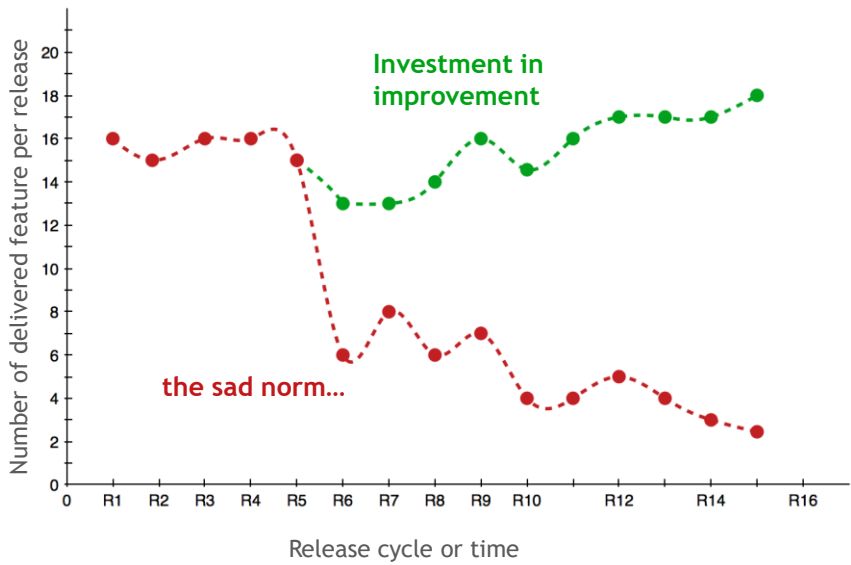


Sustainable architecture = easy to understand, to change, to extend

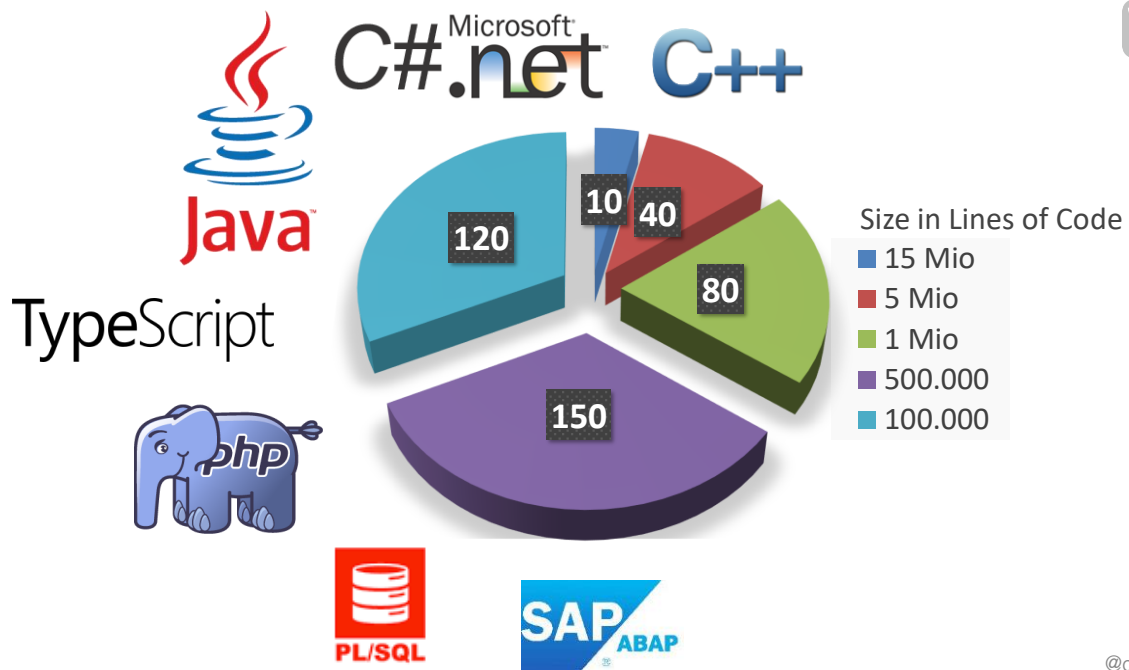
@caiolali

18

SUSTAINABLE ARCHITECTURE

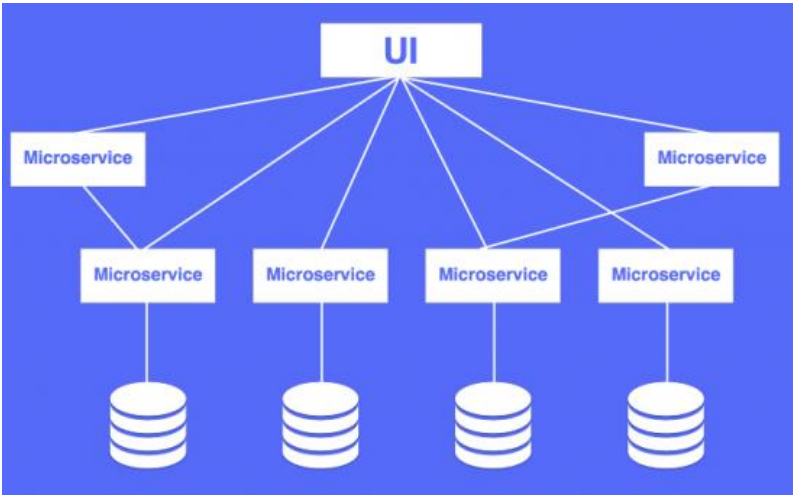


@cairolali



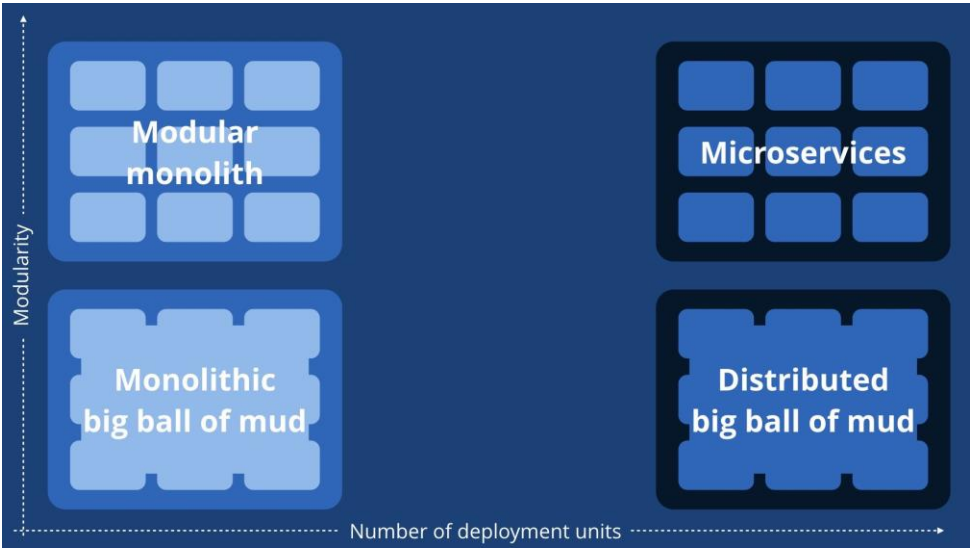
@cairolali

NEW SOLUTION: MICROSERVICES!



@cairolali

NEW SOLUTION – SAME PROBLEM



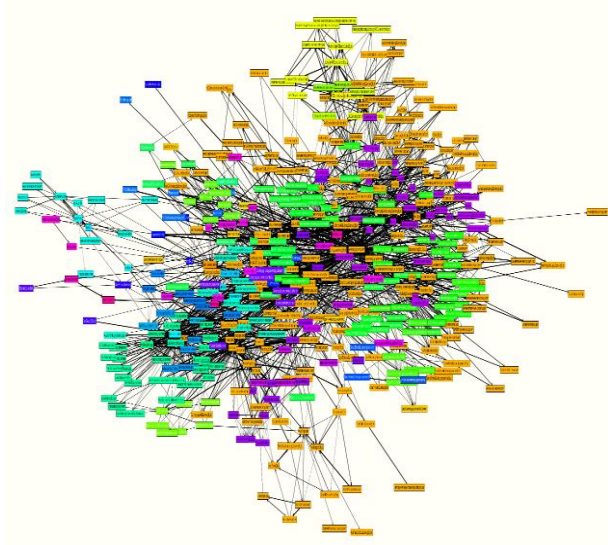
Simon Brown

@cairolali

BIG BALL OF MUD ON CLASS LEVEL



- 15 years of development
- 3 Million LOC in Java
- 463 tangled classes from 10 modules
- This cycle contains 500.000 LOC



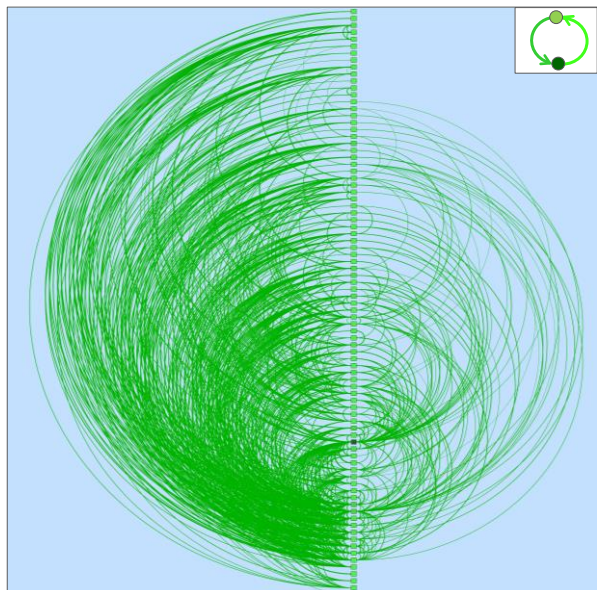
@cairolali

23

BIG BALL OF MUD ON MODULE LEVEL



- 20 years of development
- 10 Million Lines of Code in Java
- 90 Modules
- No dependency control!



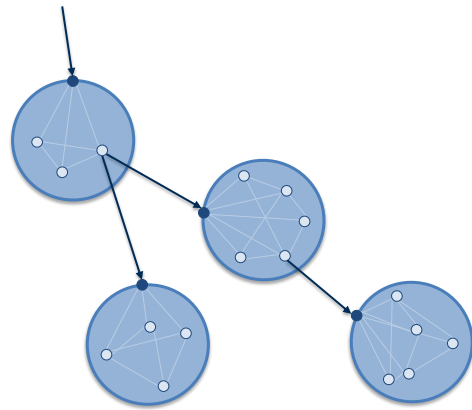
@cairolali

24

HIGH COHESION AND LOOSE COUPLING



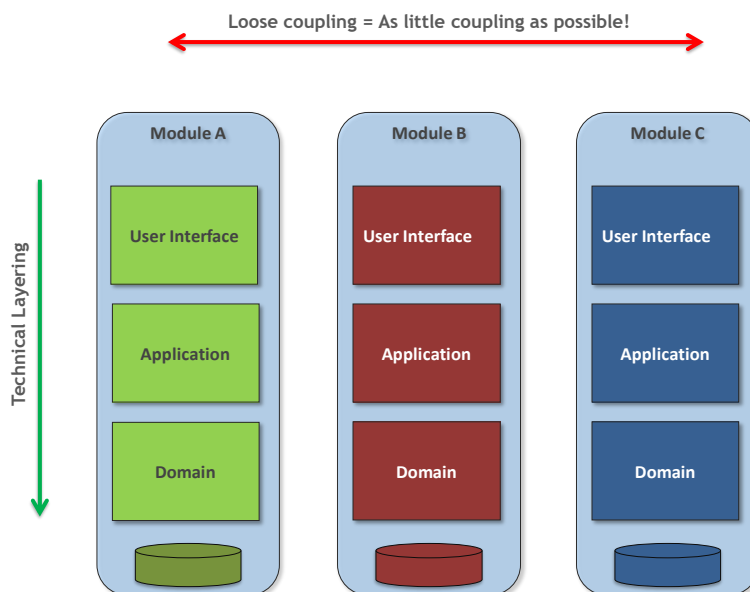
- Within a module:
 - high cohesion = strong coupling
 - Single Responsibility Principle
 - Separation of concern
- Outside a module:
 - loose coupling = low coupling
 - Information hiding
 - Separation of concern



@cairolali

25

HIGH COHERSION AND LOW COUPLING FOR MODUES



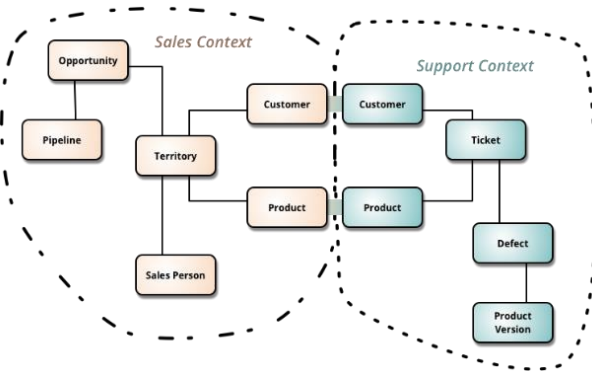
@cairolali

26

THE NEED OF DOMAIN ARCHITECTURE

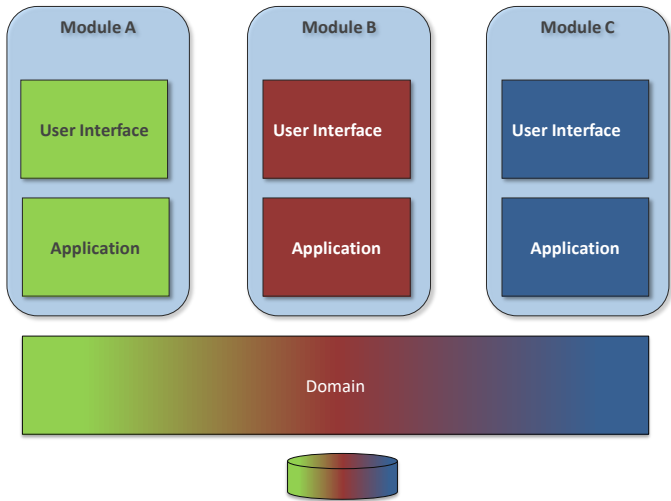


An architecture (with or without microservices) only then has **high cohesion** and **low coupling** if it follows a **domain architecture**!



@cairolali

ONE DOMAIN MODEL AND DATABASE FOR ALL



@cairolali

THE OVERALL CONSISTENT MODEL



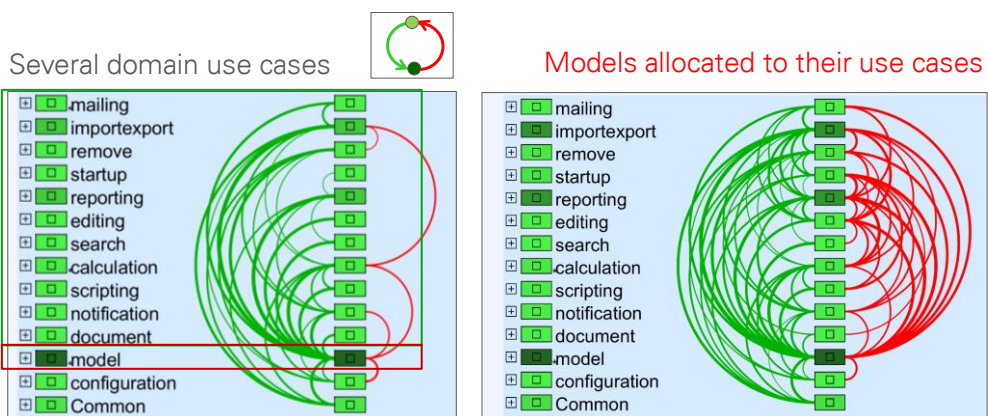
- In the 1990th fruitless tried on data models
 - The idea was to create **one model** for an entire enterprise or even an entire domain
- Much too complicated
- High degree of coordination between teams
- Often ends up with a lot of coupling in a big ball of mud



@cairolali

29

ONE SHARED MODEL



@cairolali

30

MODELS IN MICROSERVICES

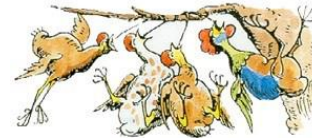


→ domains need to be divided into **subdomains** with **small models**

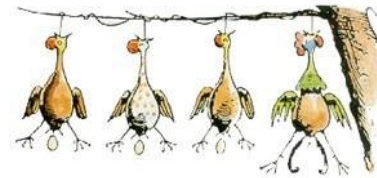


▪ Separate models

- Allow **independent changes (separation of concern)**
- have to represent a smaller set of the **requirements (high cohesion)**
- Need a clear **boarder (loose coupling)**

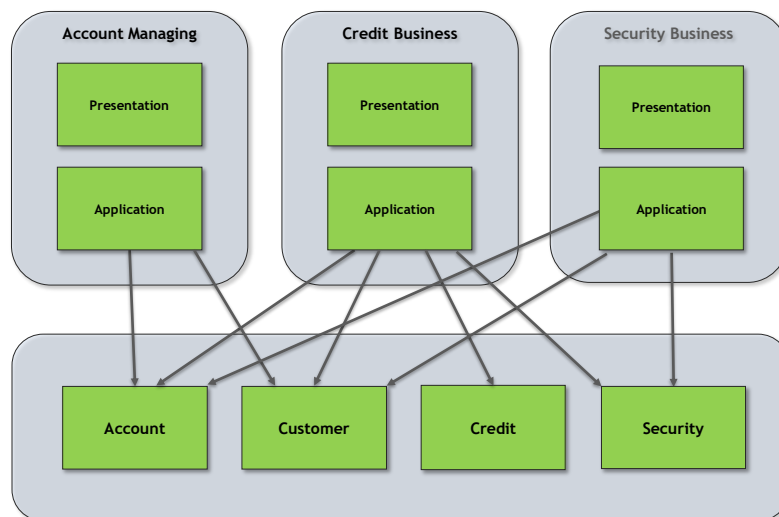


→ Each model and its subdomain should be **small enough** to be programmed by **one team**.



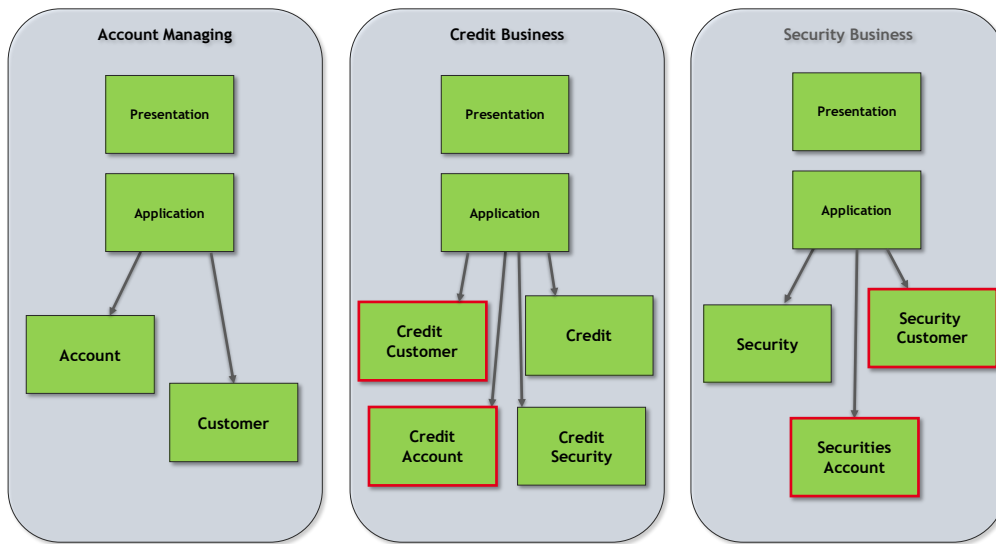
@cairolali

31



@cairolali

32

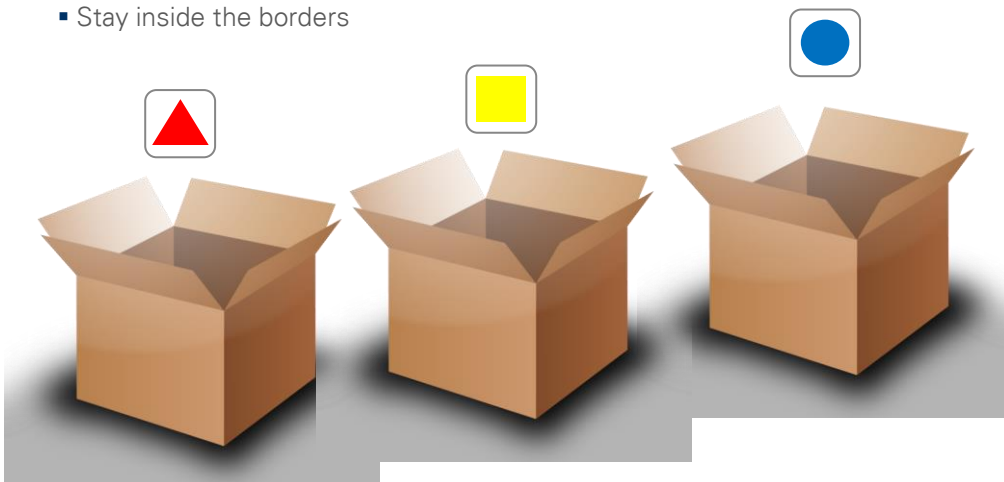


@cairolali

33

EACH MODULE HAS ITS OWN MODEL

- Each team can work freely on their own domain model
 - Know the limits
 - Stay inside the borders



@cairolali

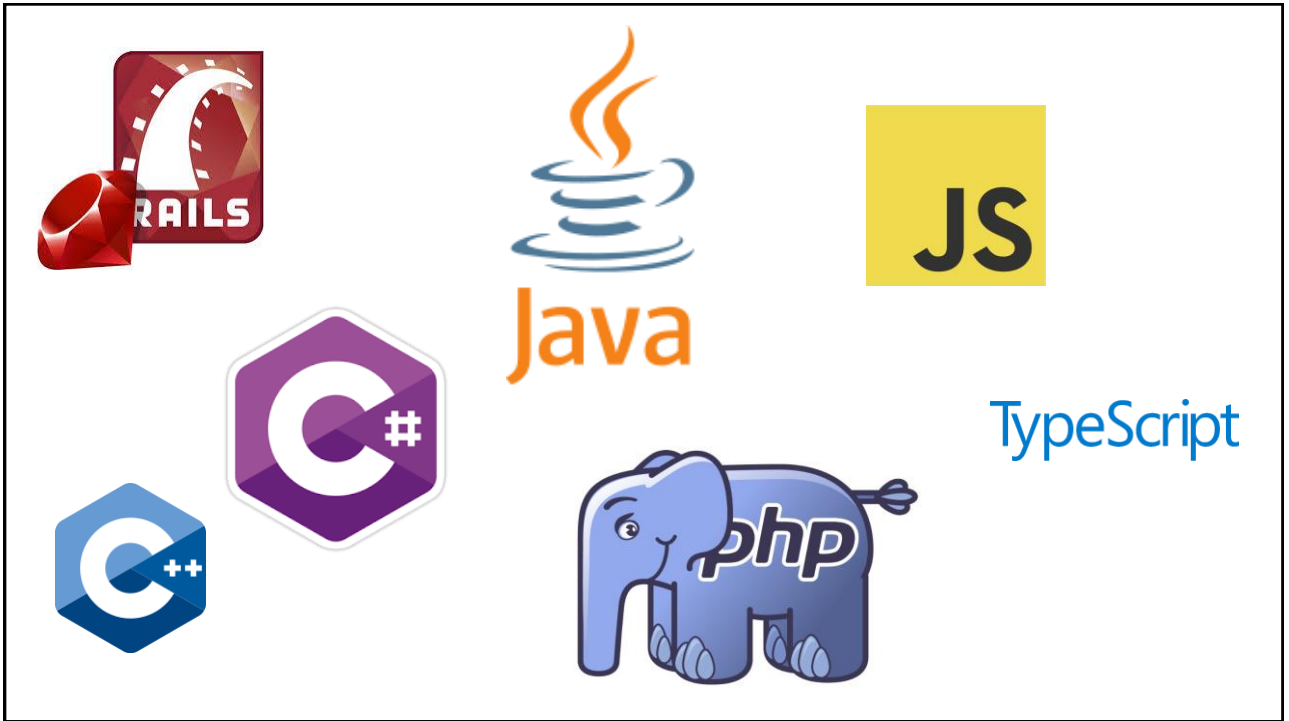
34



35



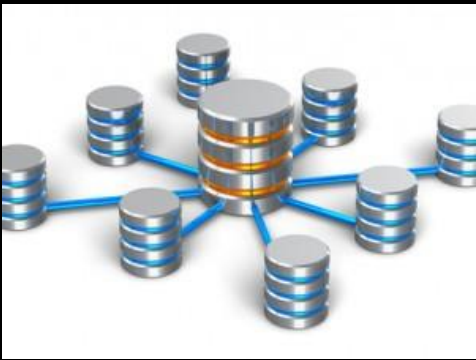
36

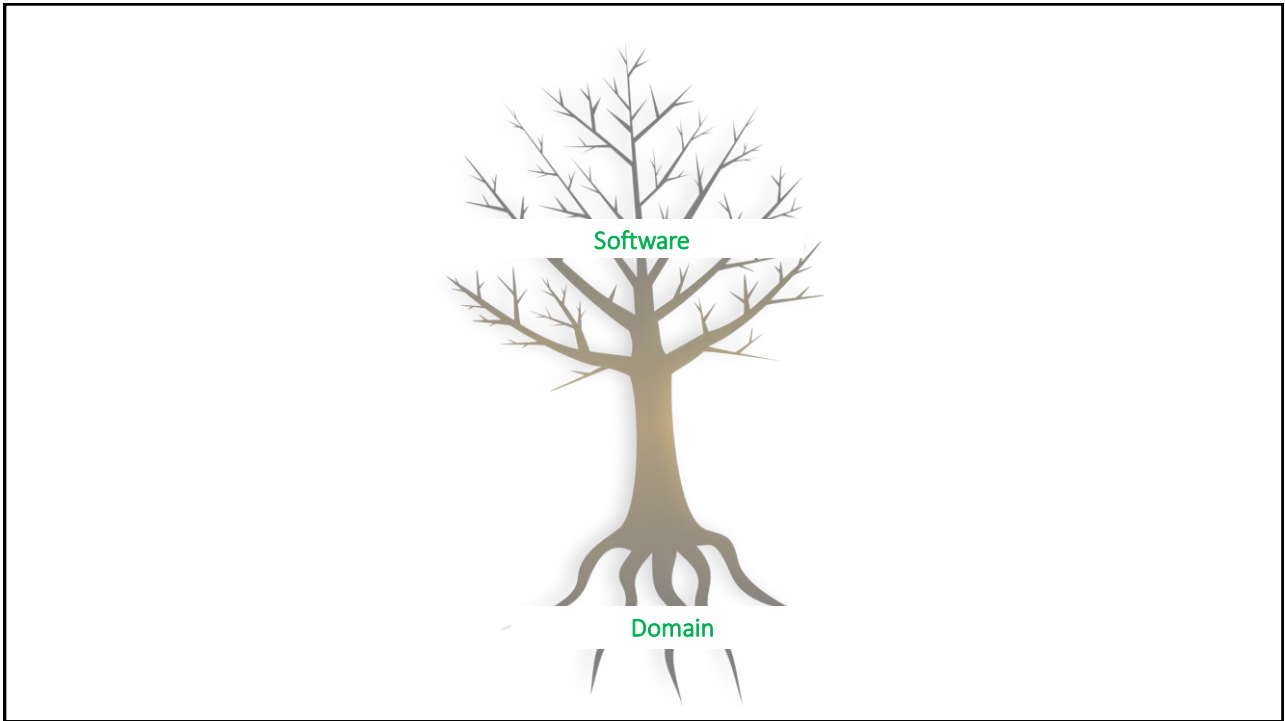


37



38





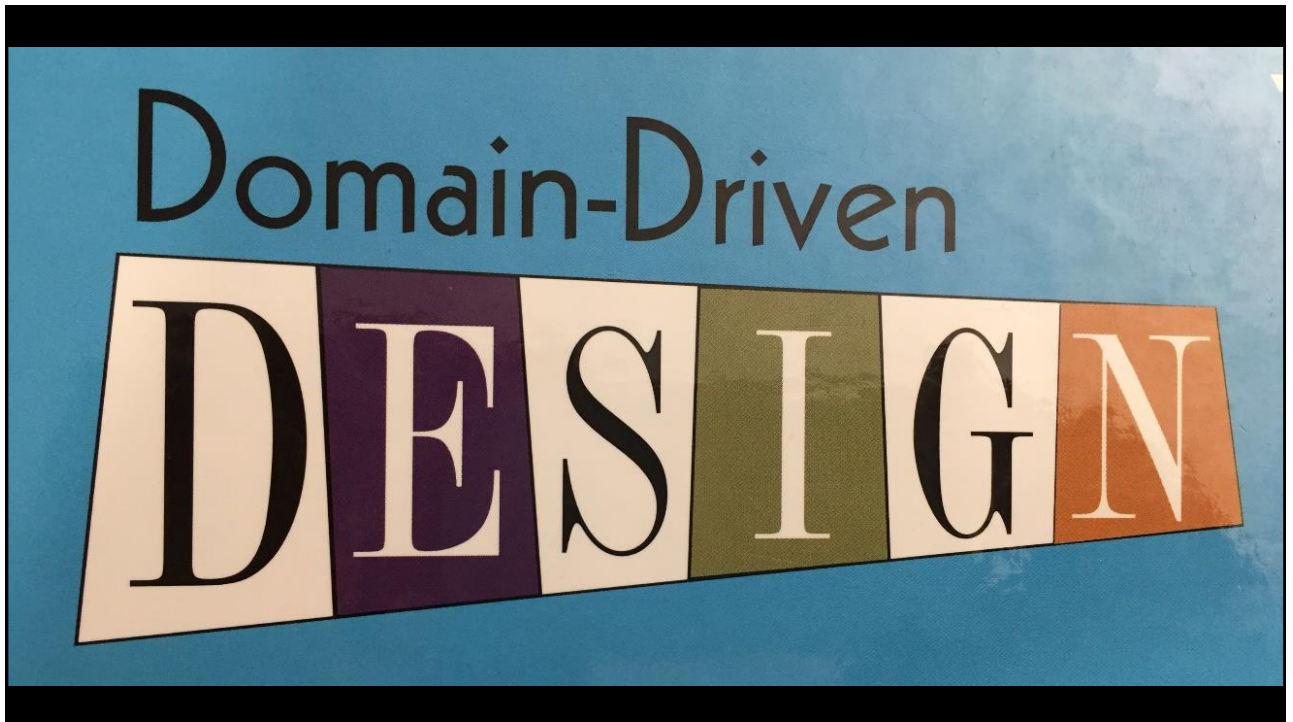
41



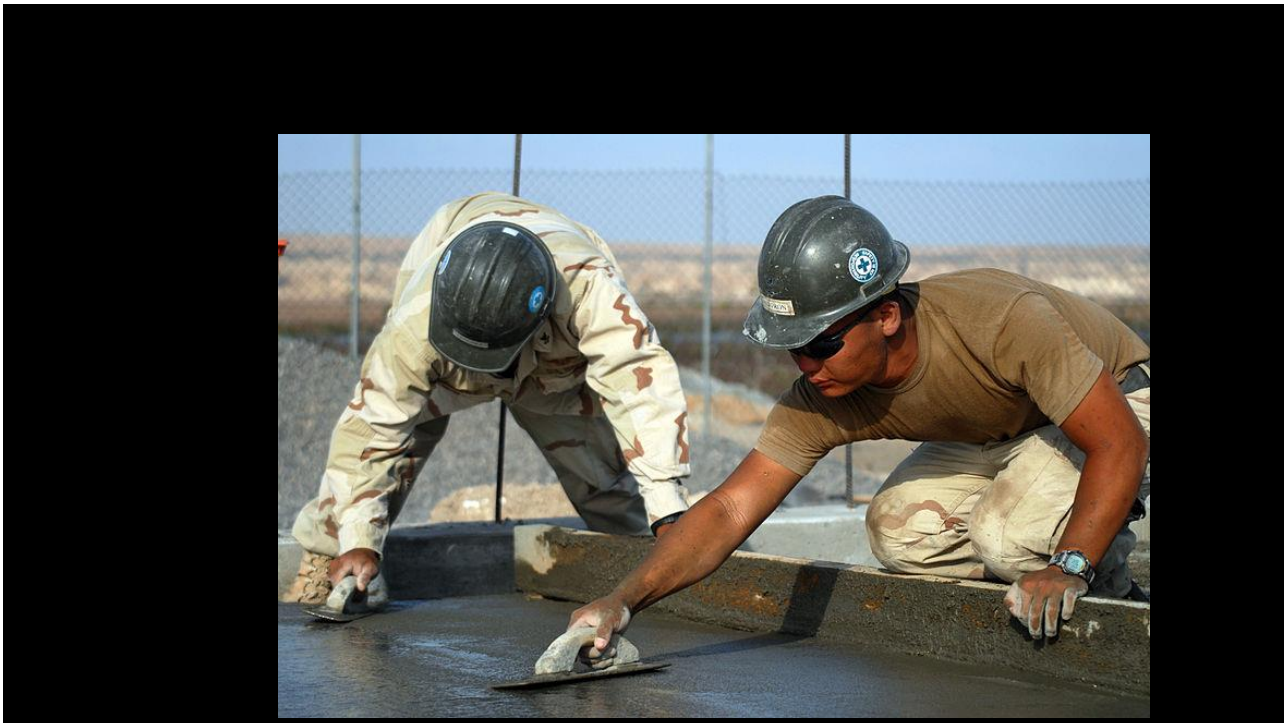
42

The source code only contains the **understanding** of the developer, not the requirements written down in the **backlog** by the business analysts.

43



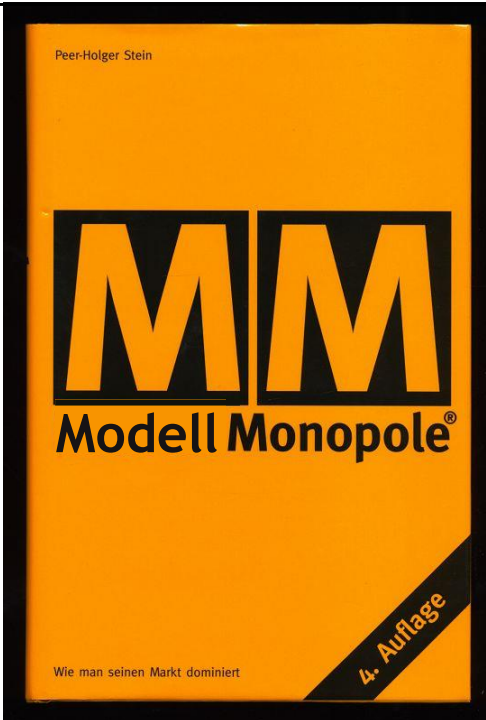
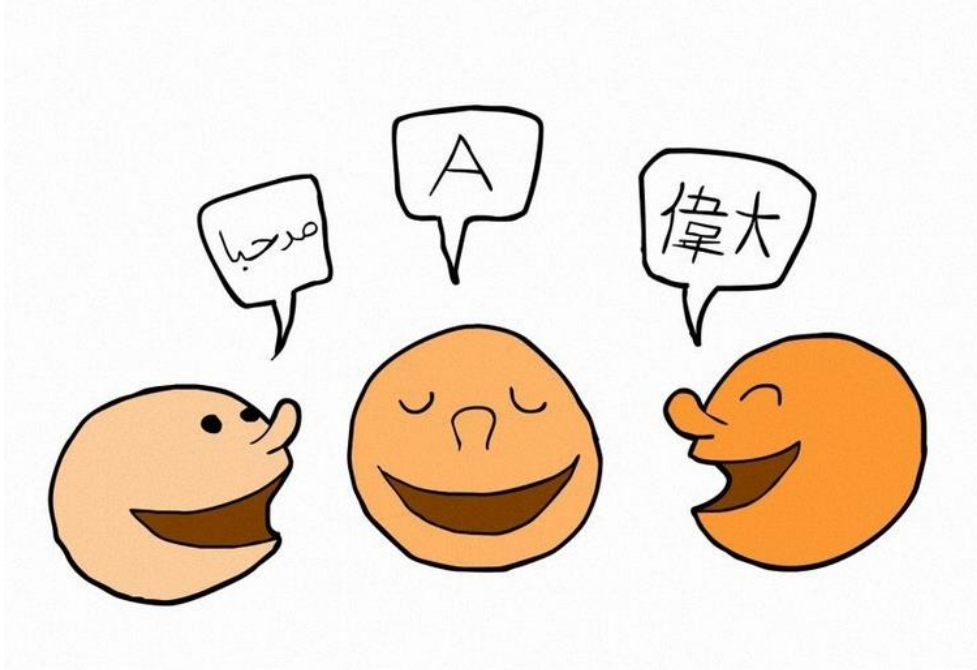
44

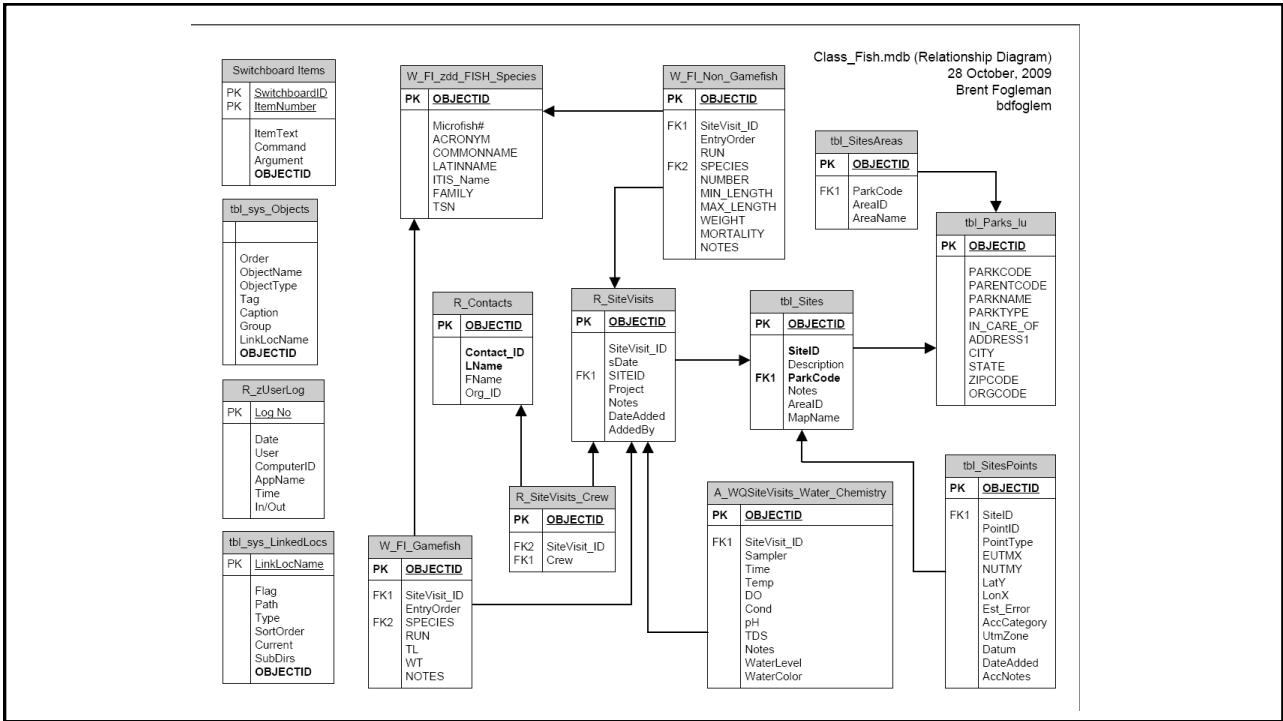


45



46







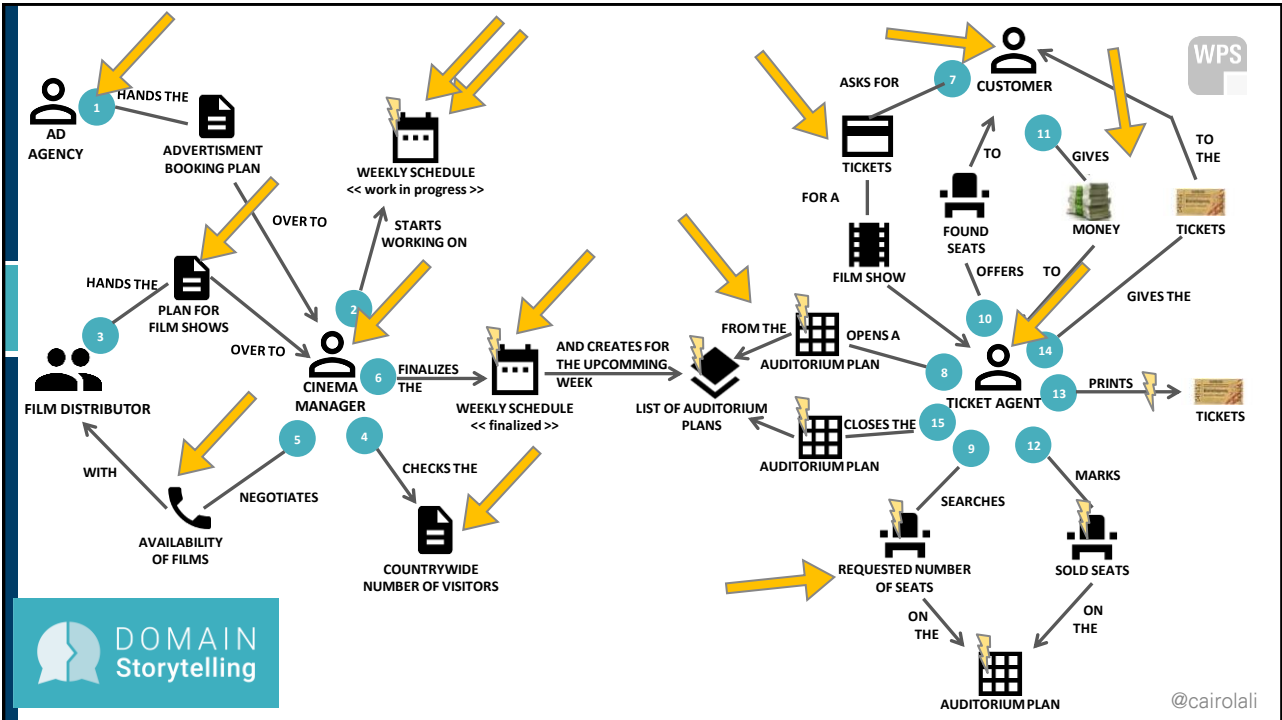
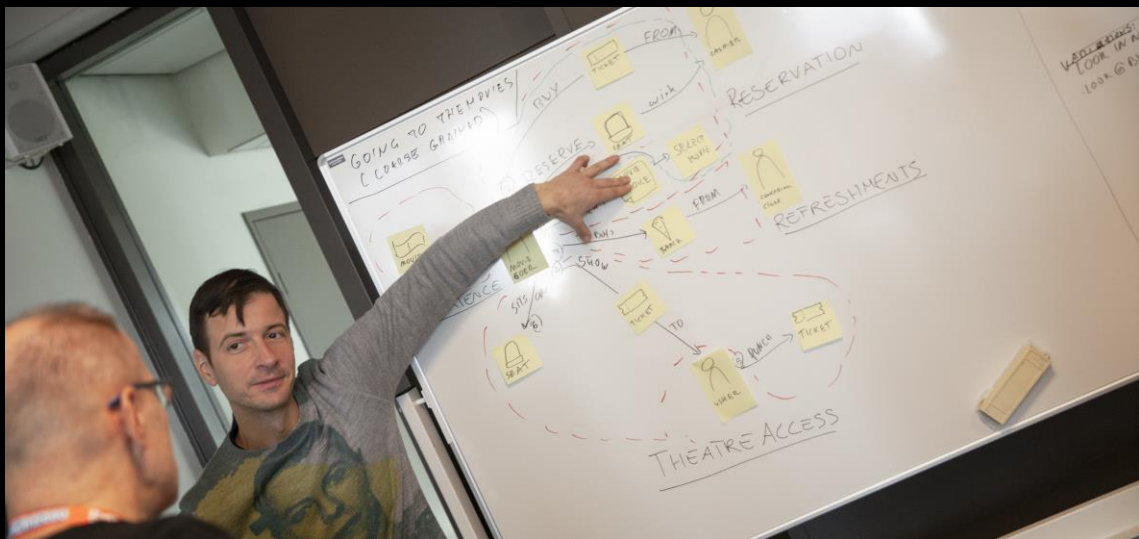
51

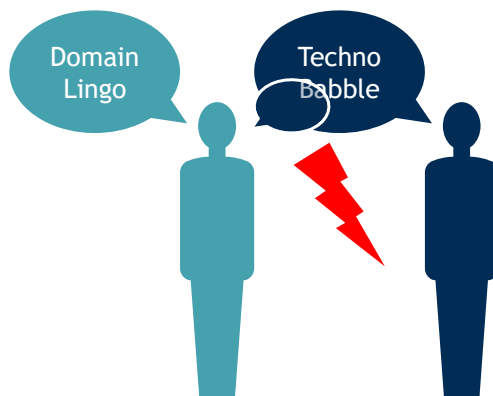
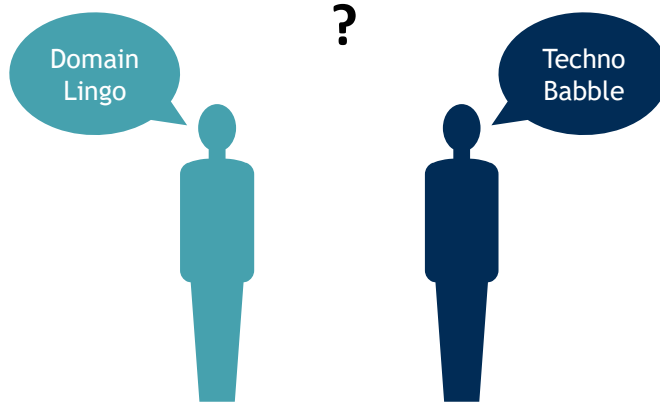


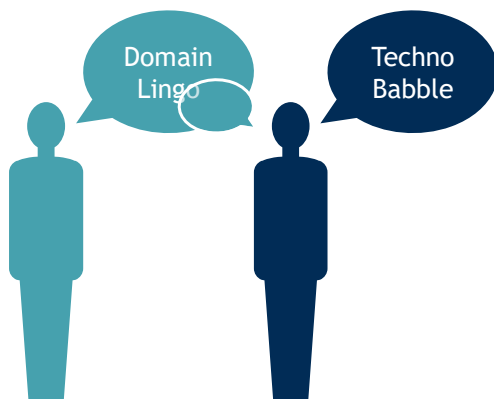
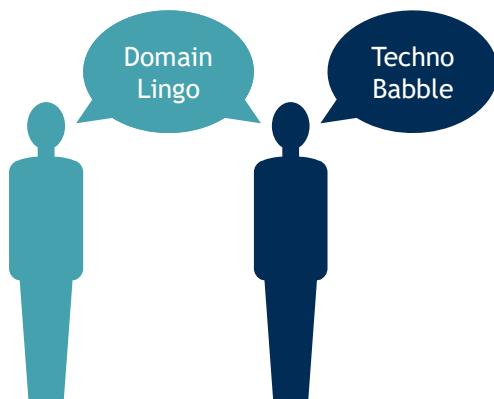
52

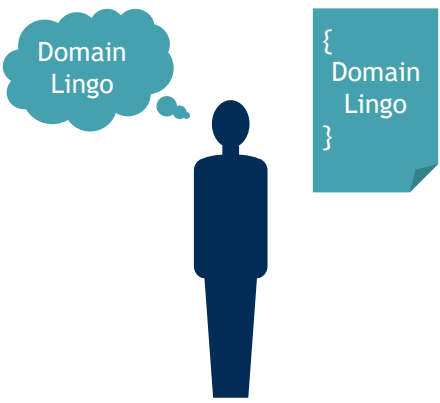
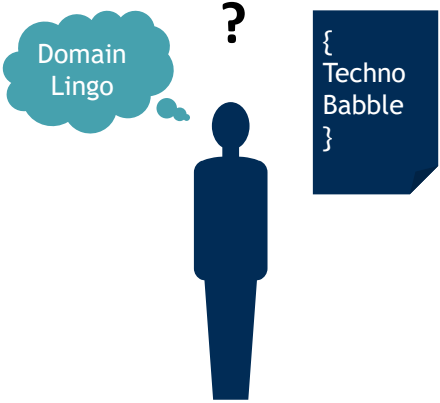
Domain Storytelling

<https://domainstorytelling.org/>





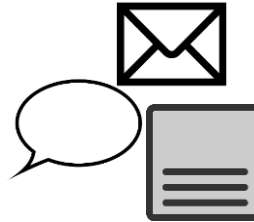




UBIQUITOUS LANGUAGE



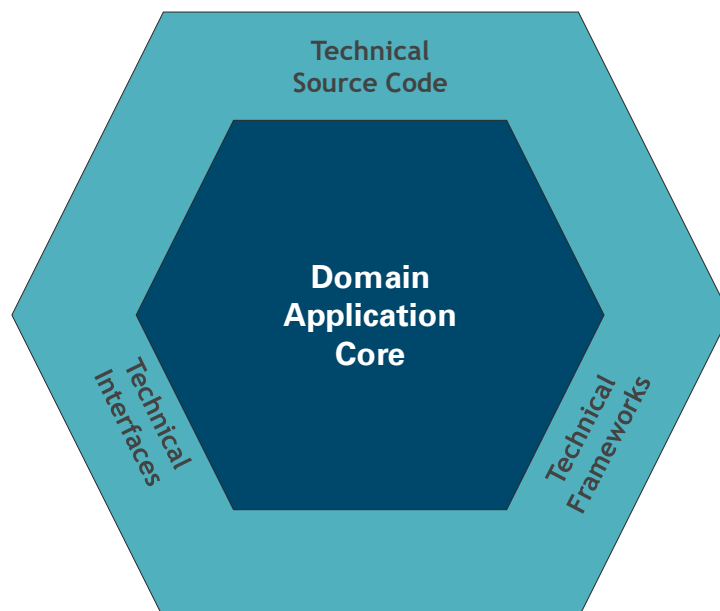
- Use the common language in
 - All **communications**
 - Speech
 - Writing
 - Diagrams
 - In the **code**



- **Basically everywhere** → that is why it is called a *ubiquitous* language

@cairolali

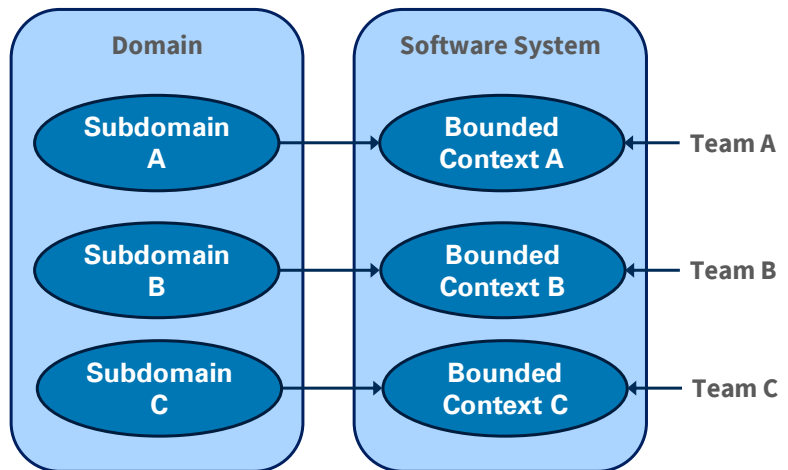
61



@cairolali

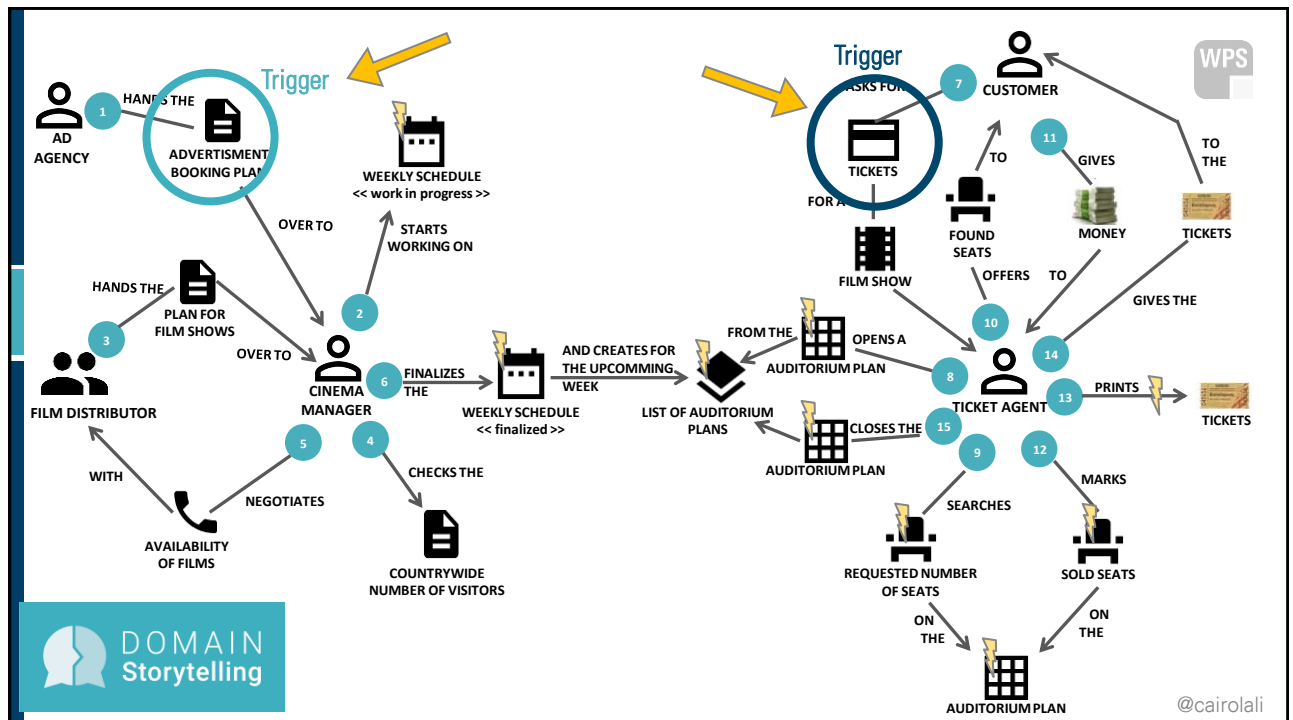
62

- Find the **subdomains** in the domain and decompose your software accordingly.
- Set explicit **boundaries** with the bounded contexts for
 - domain-oriented **parts** in the **software** and
 - in the **team organization**
- Goal: high **cohesion** + loose **coupling**



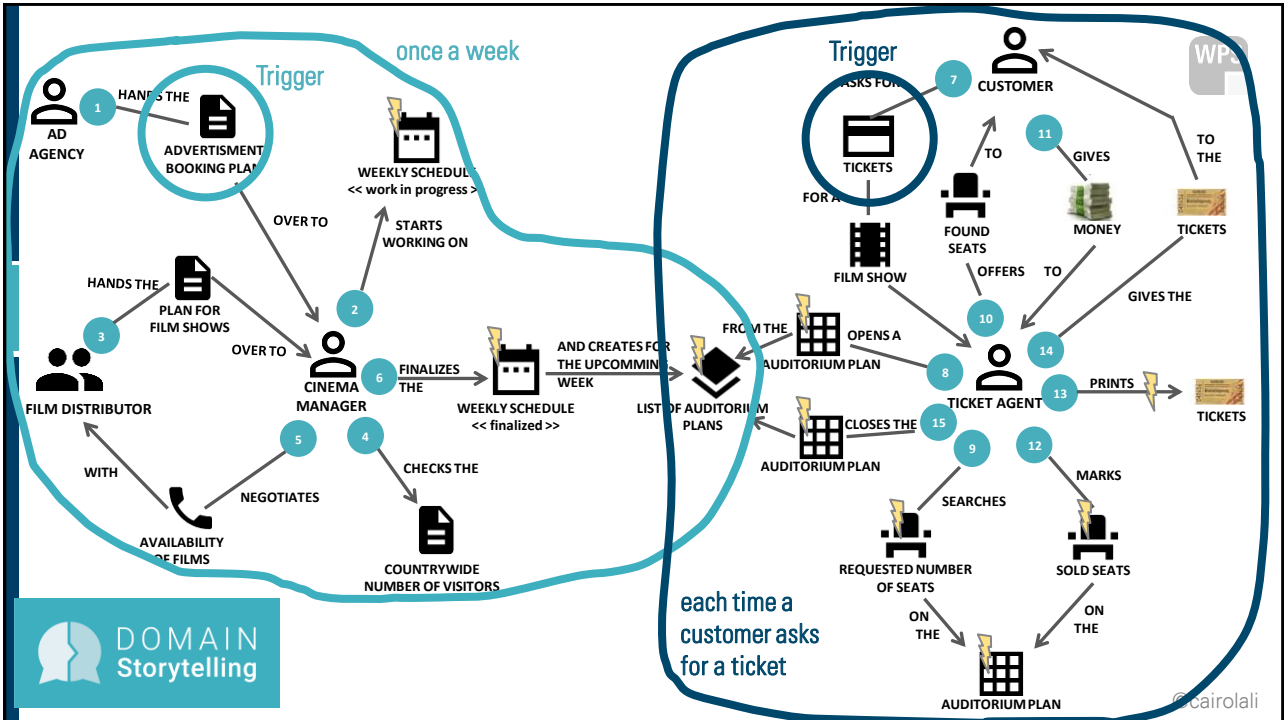
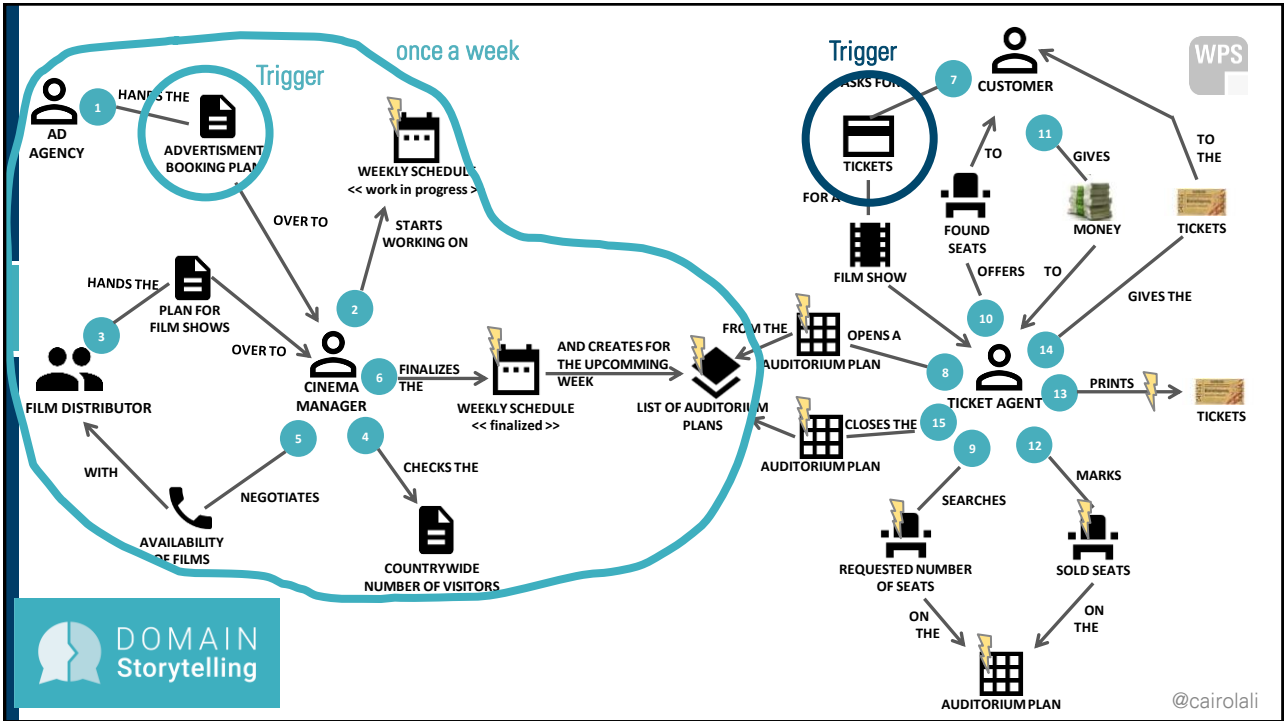
@caiolali

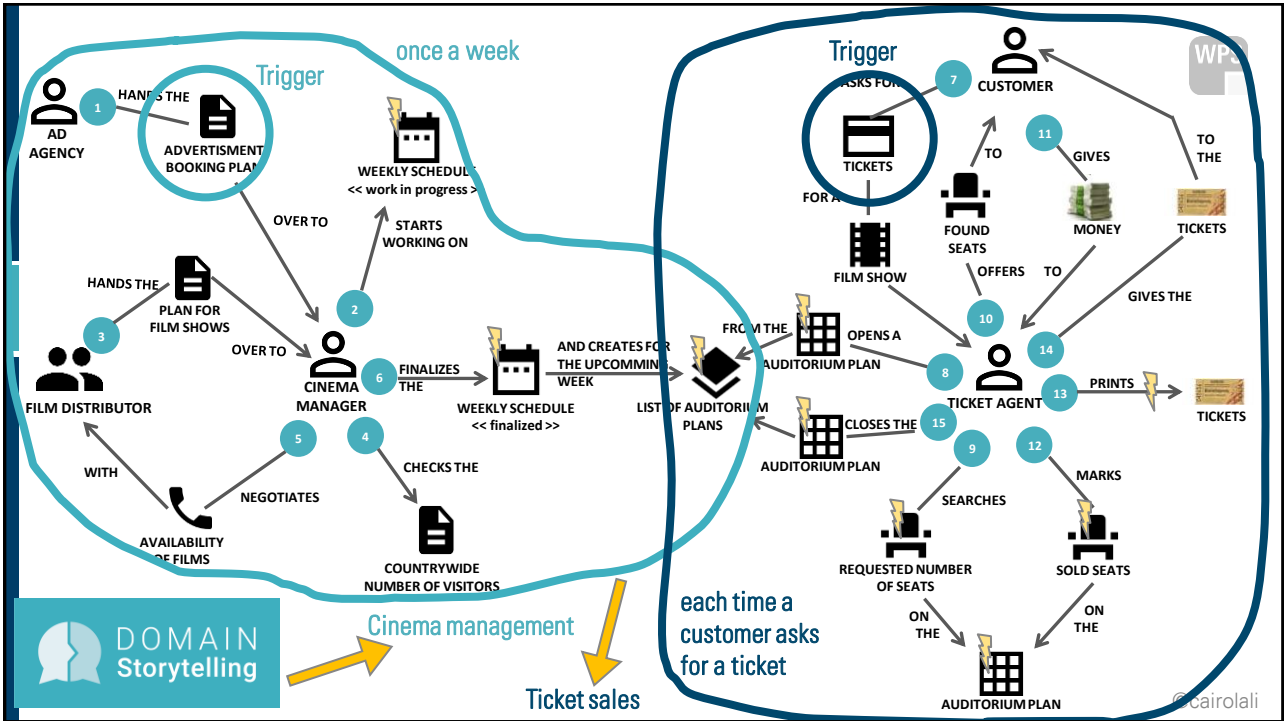
63

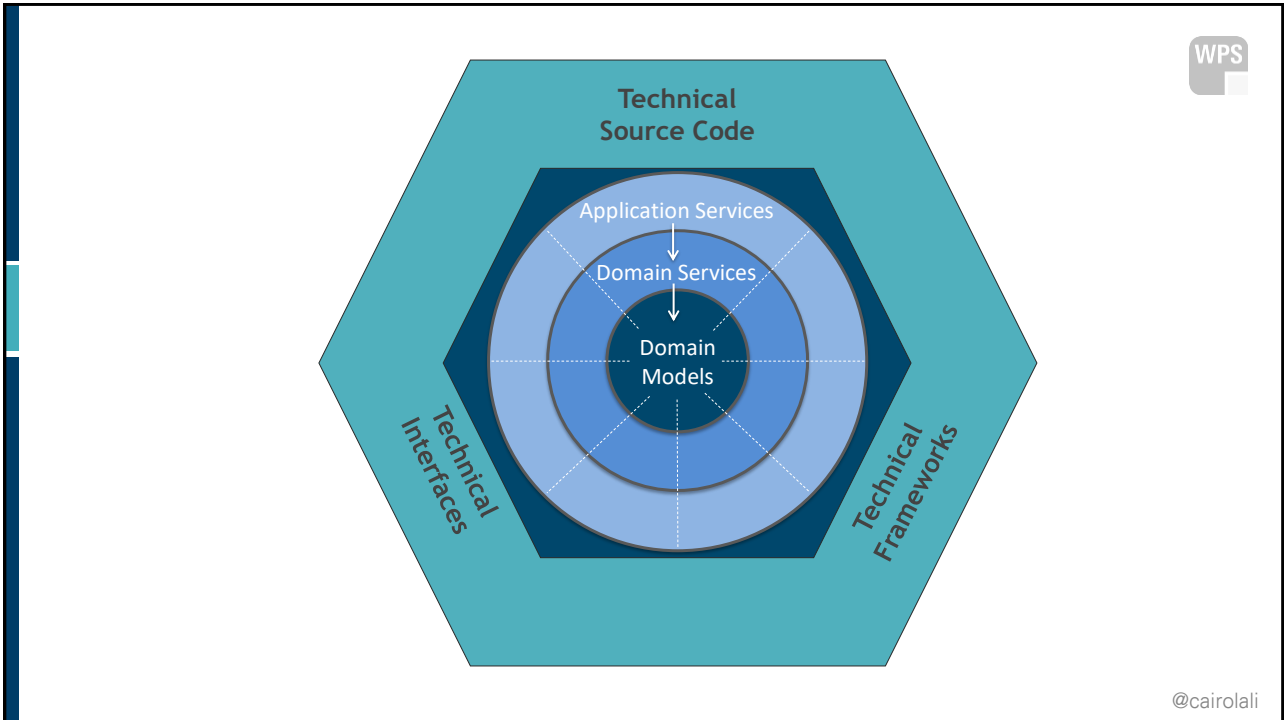
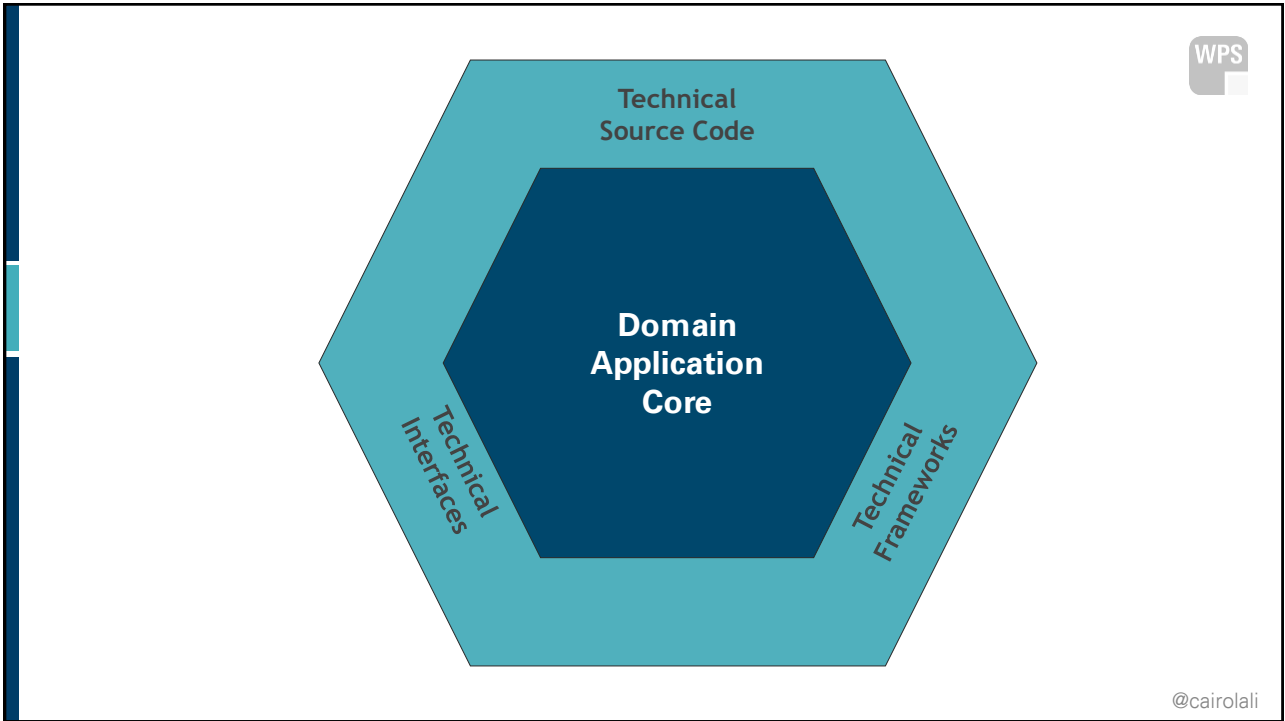


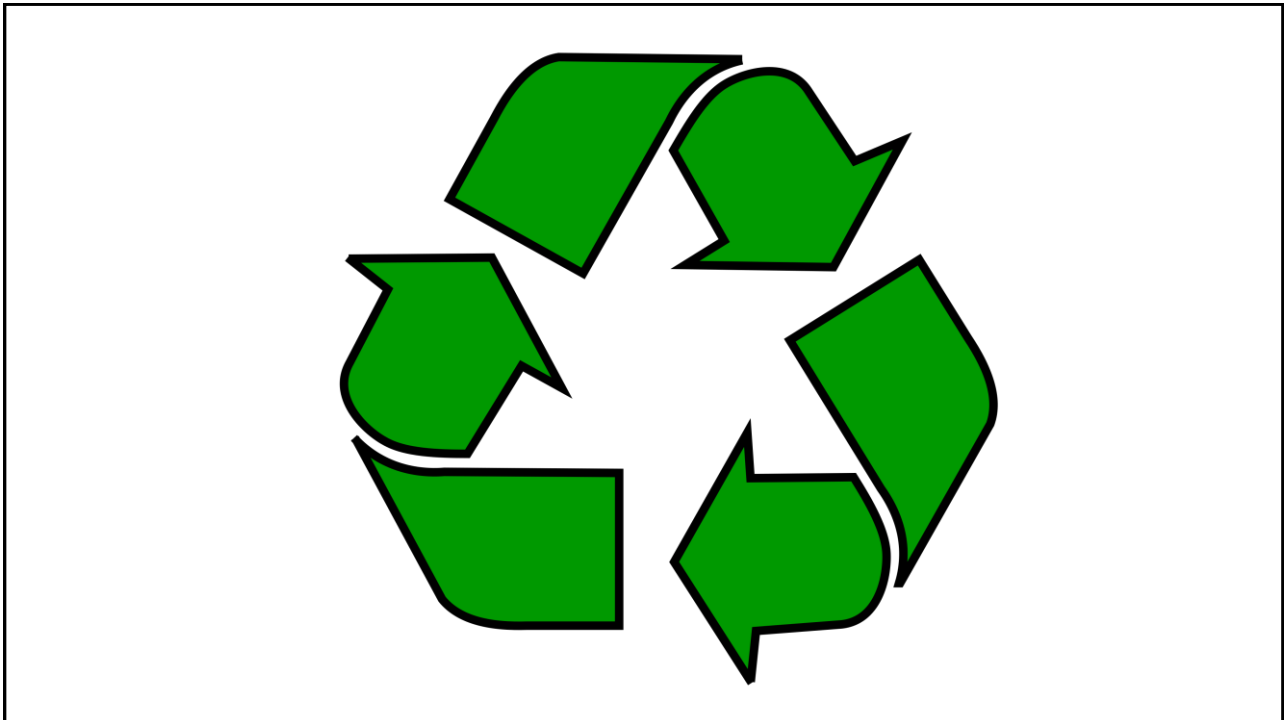
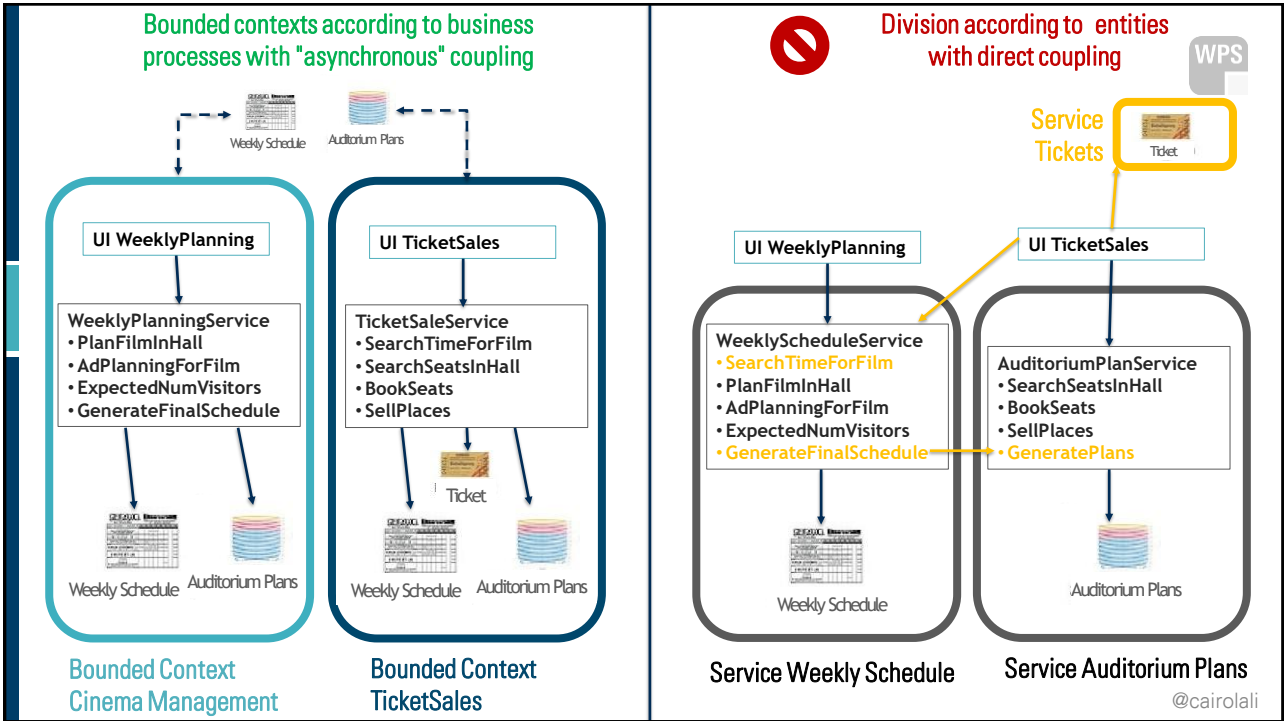
@caiolali

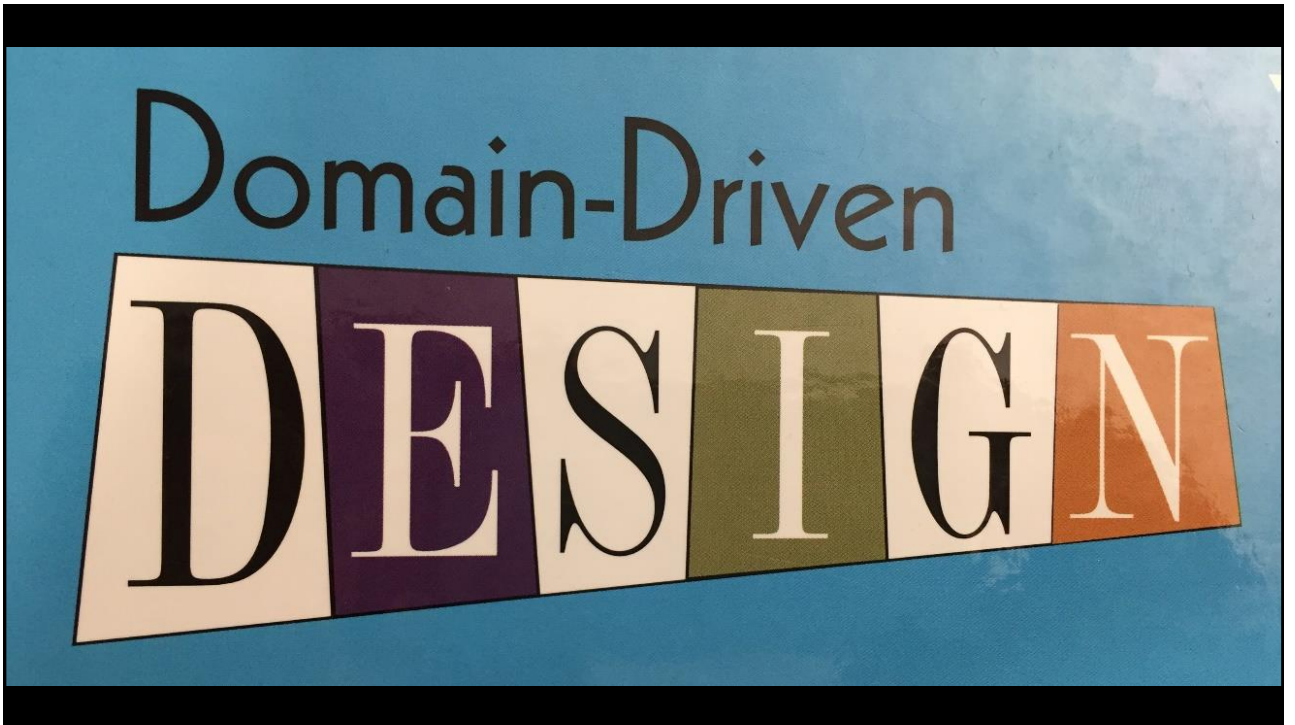
64



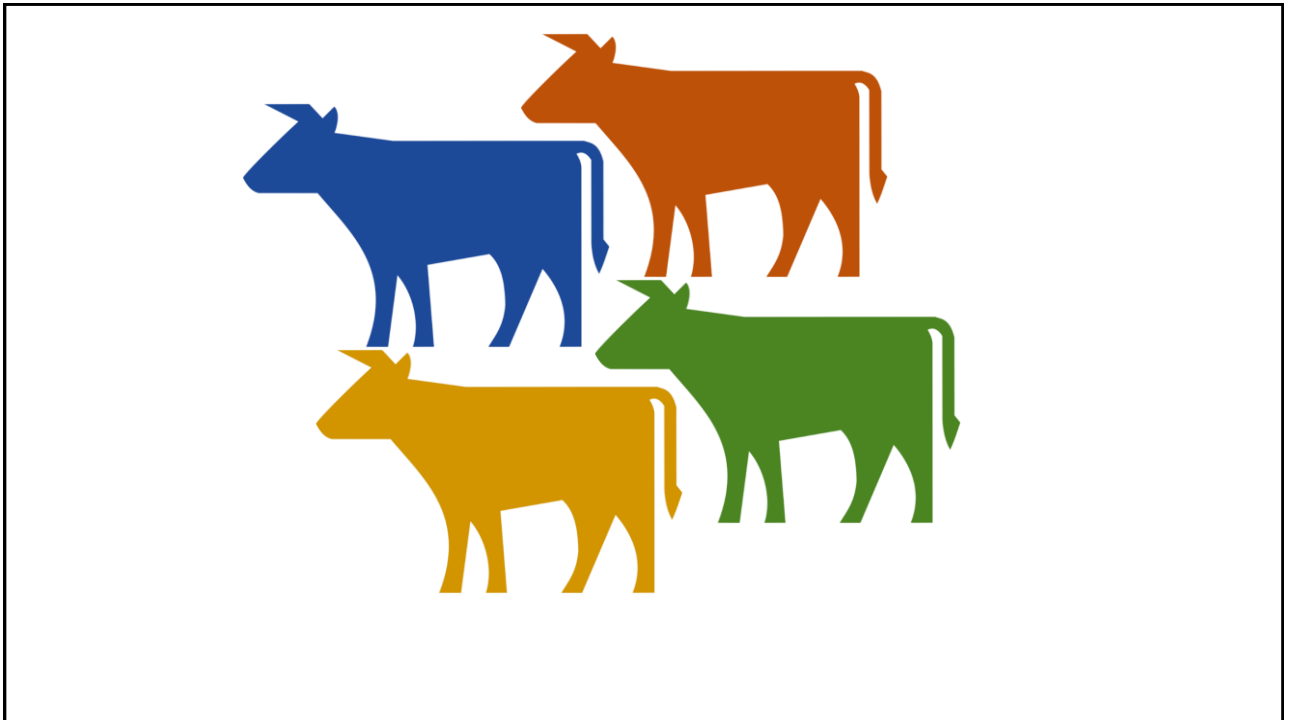




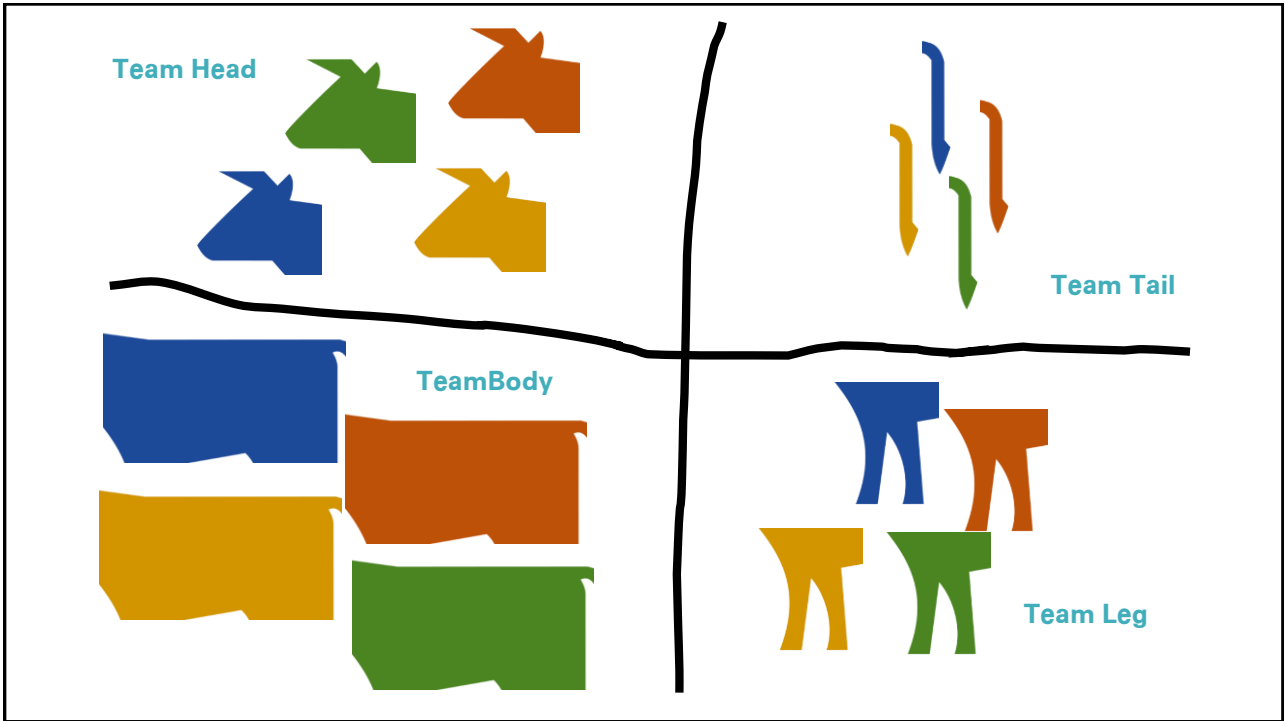




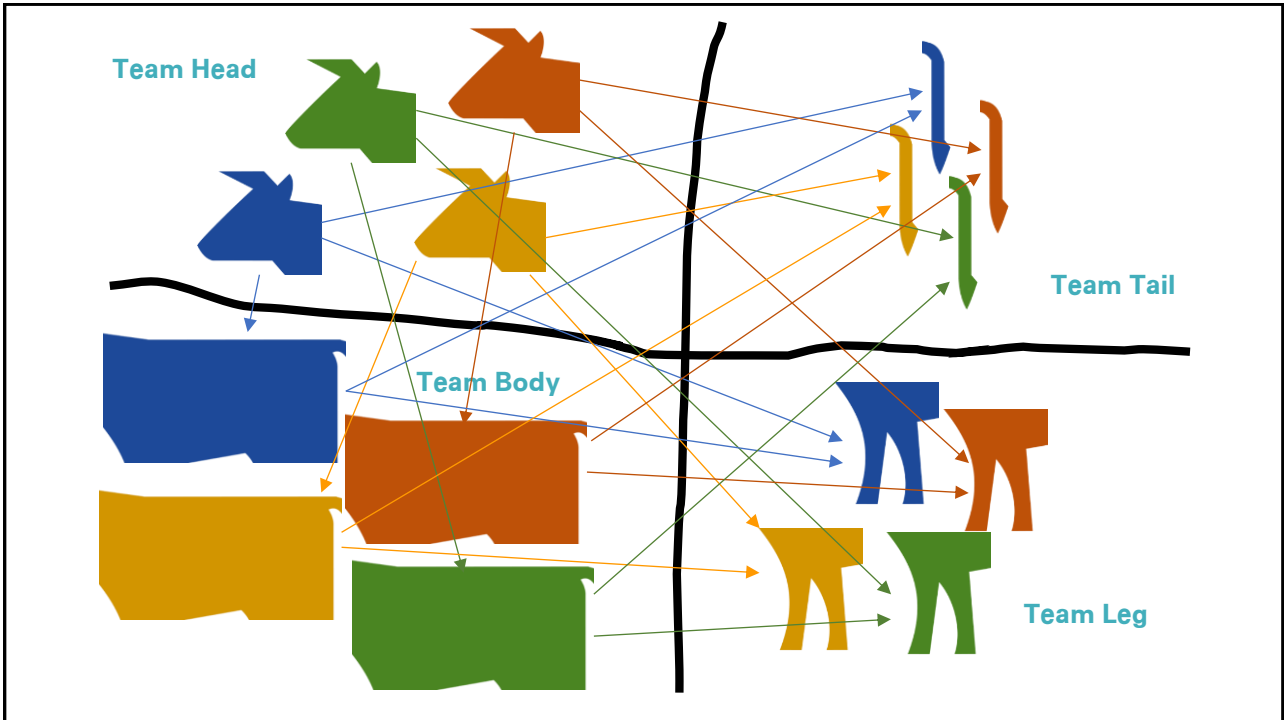
75



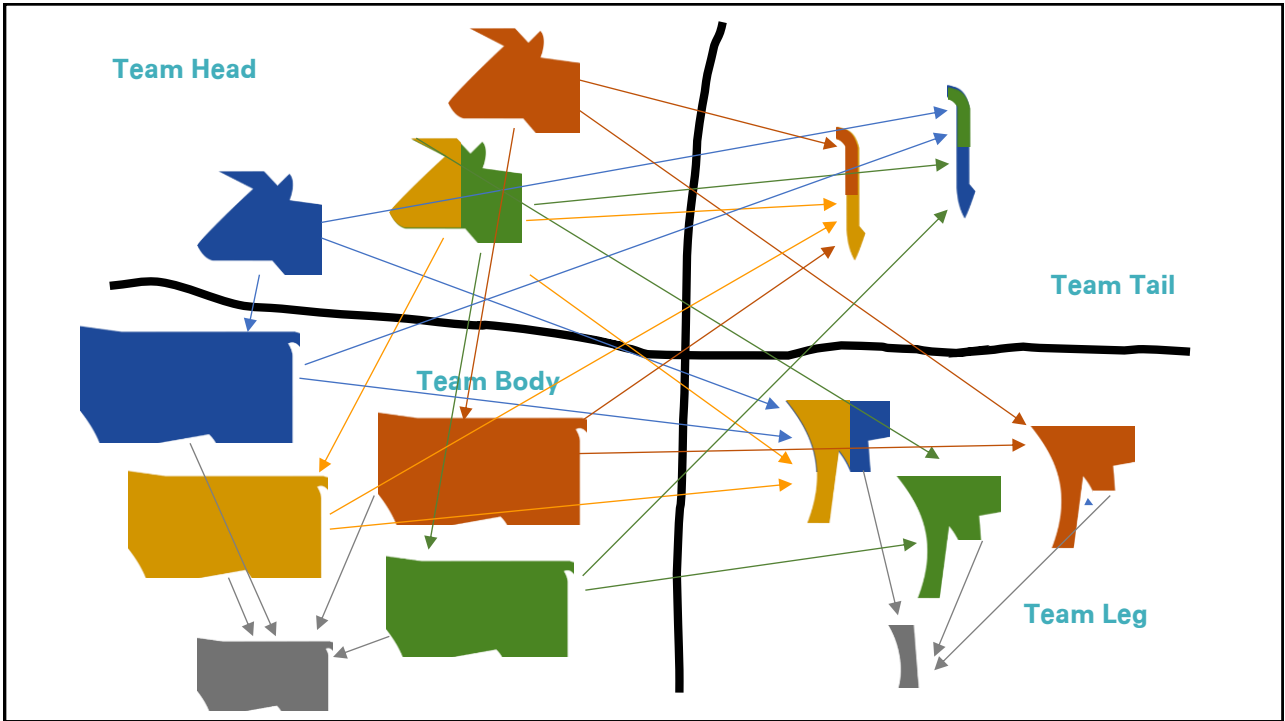
76



77



78

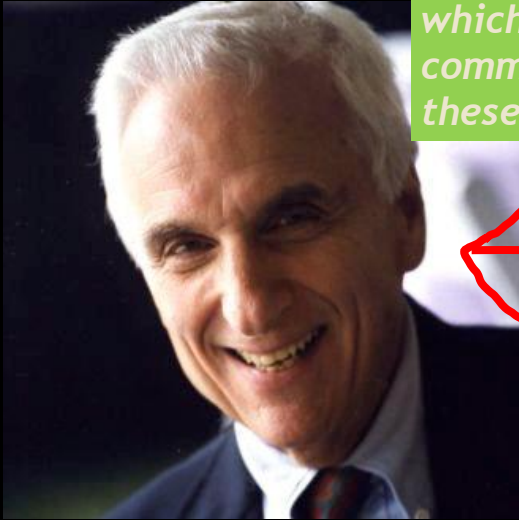


79



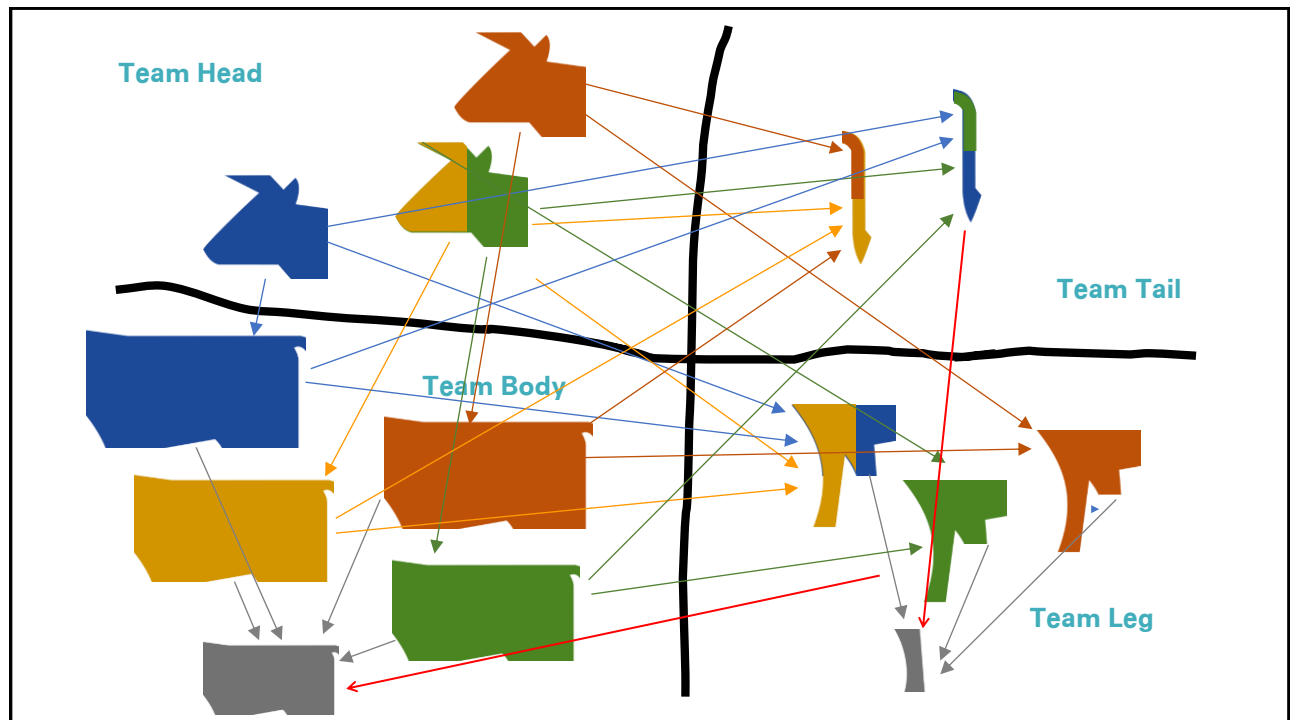
80

“Organizations which design systems are constrained to produce designs which are copies of the communication structures of these organizations.”

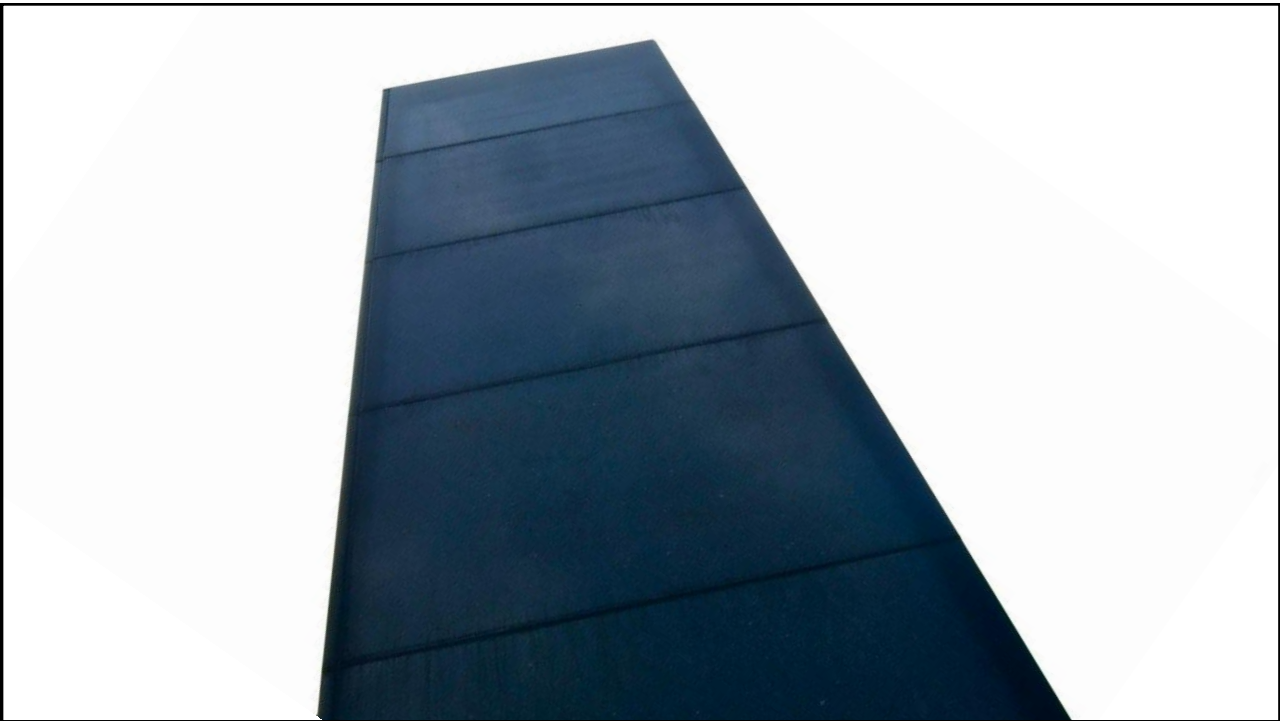


Melvin Conway

81



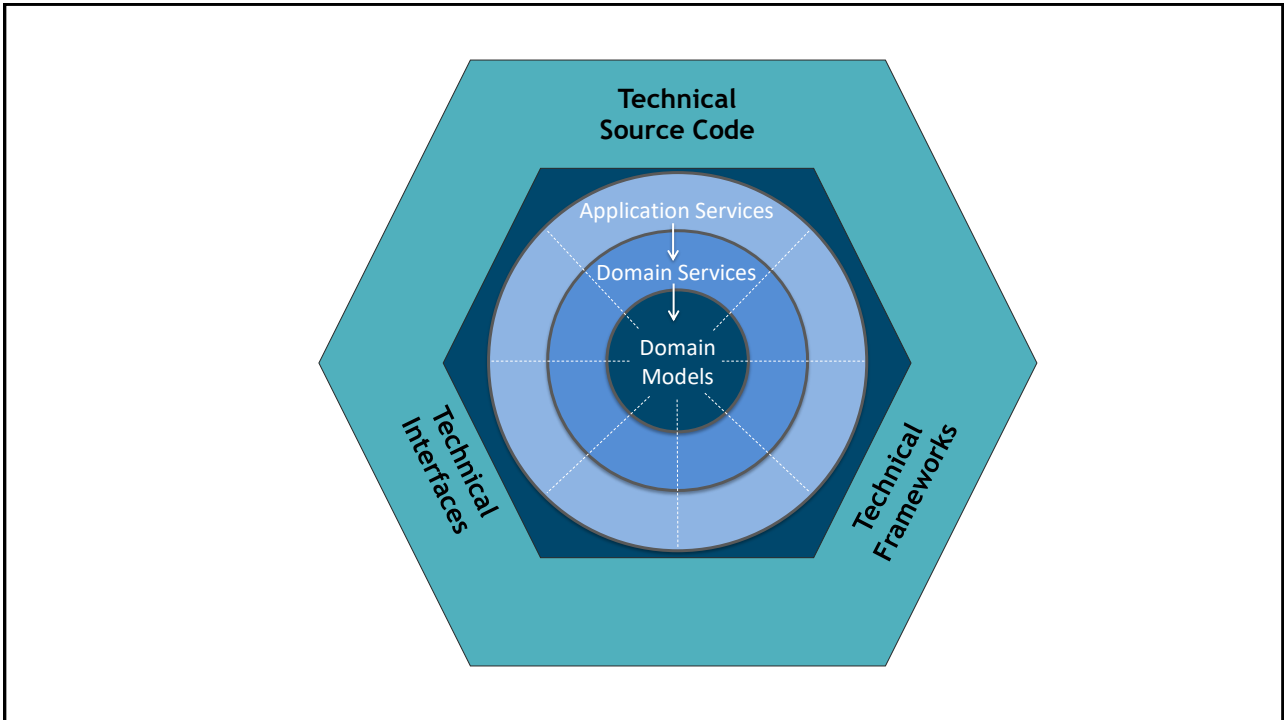
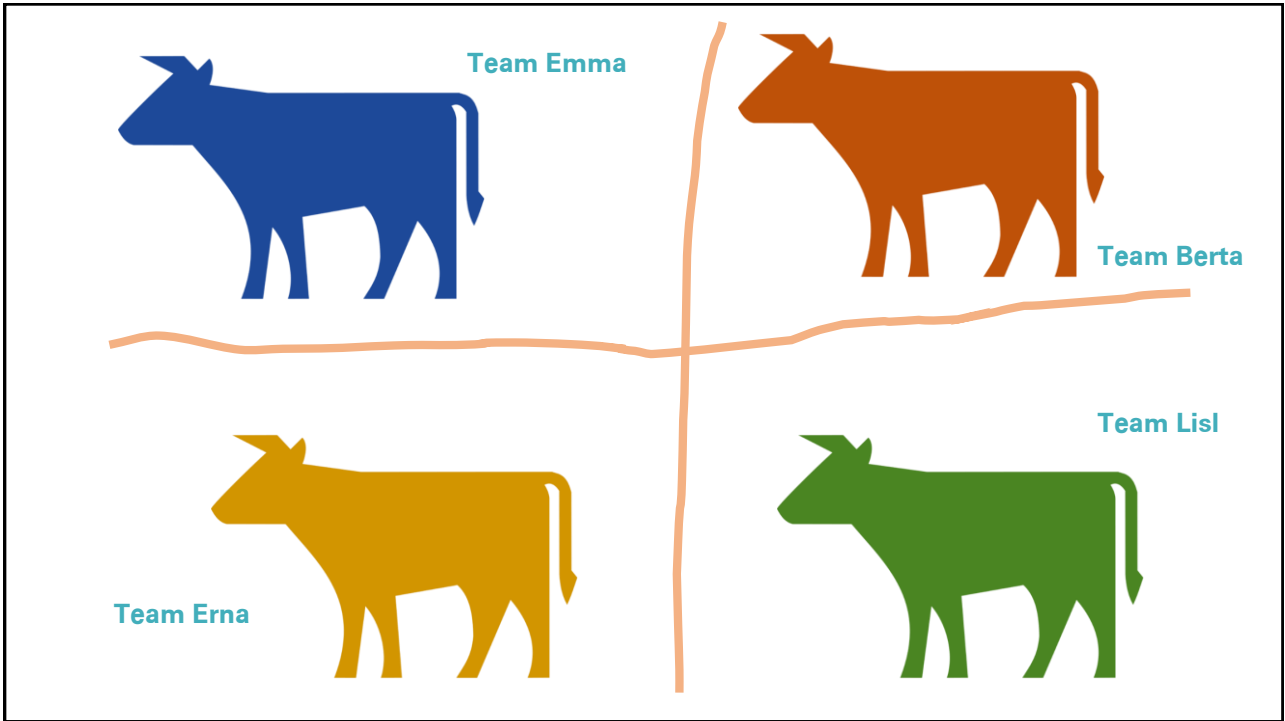
82

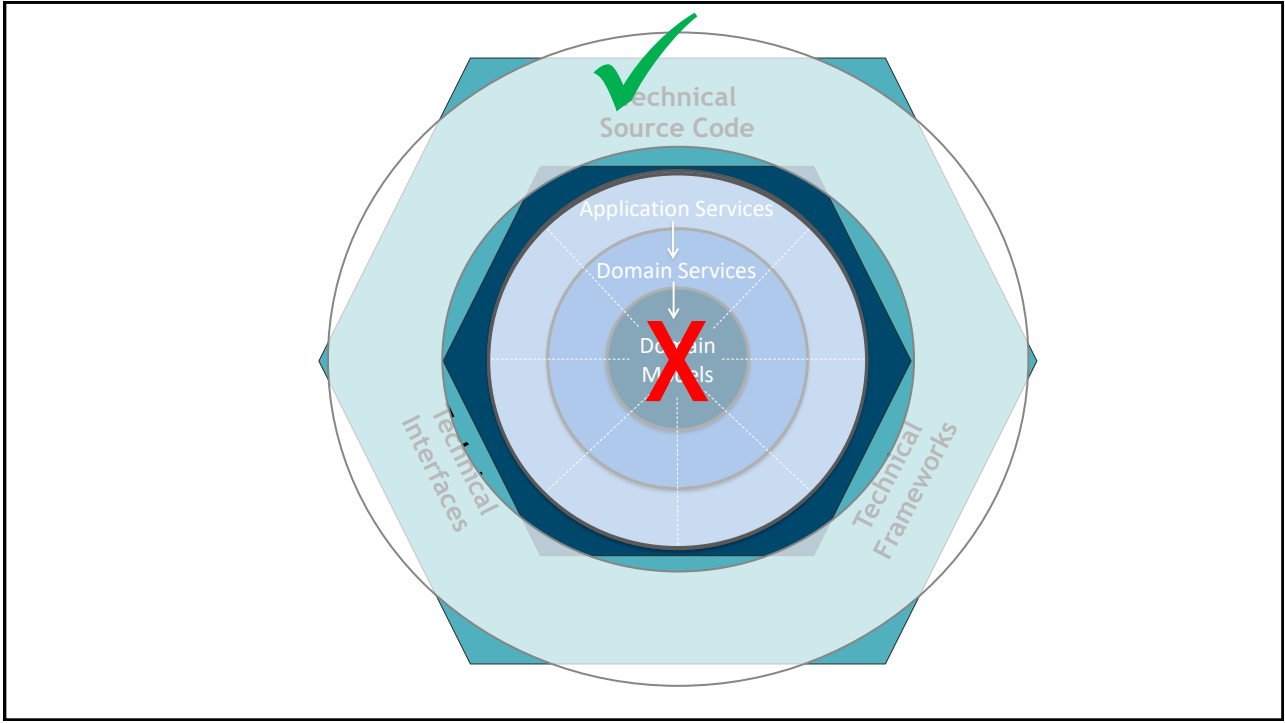
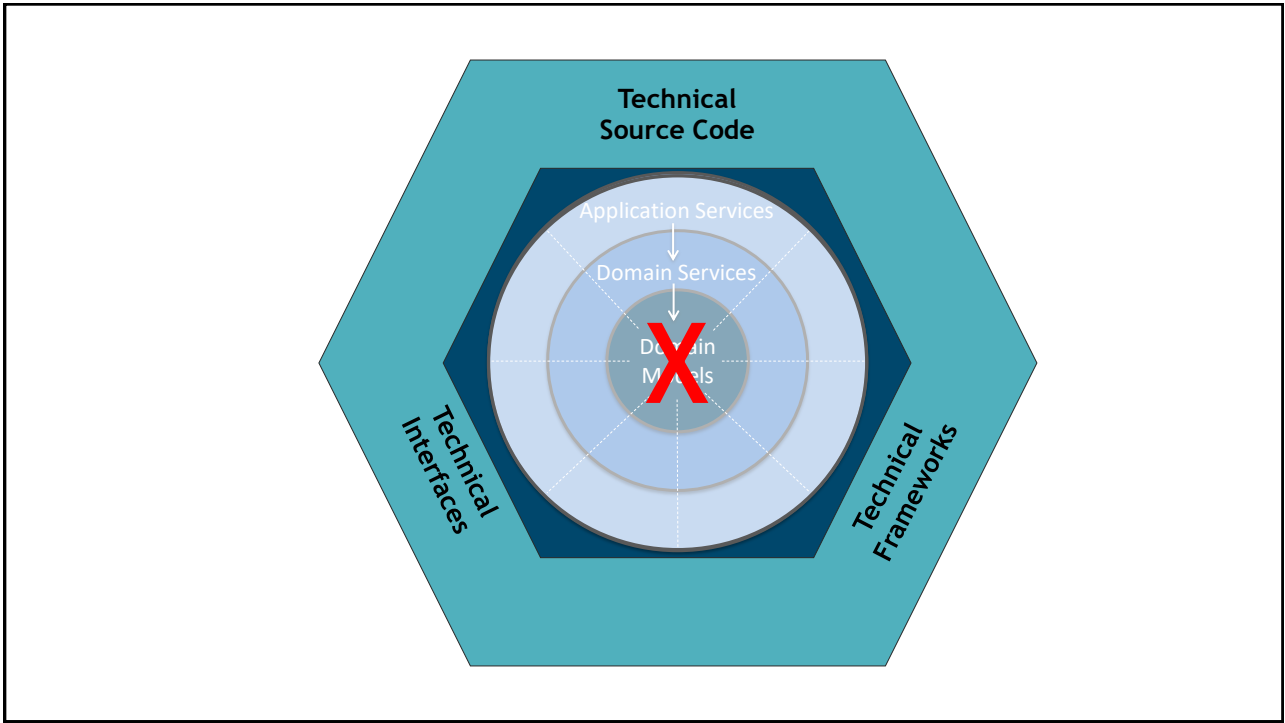


83

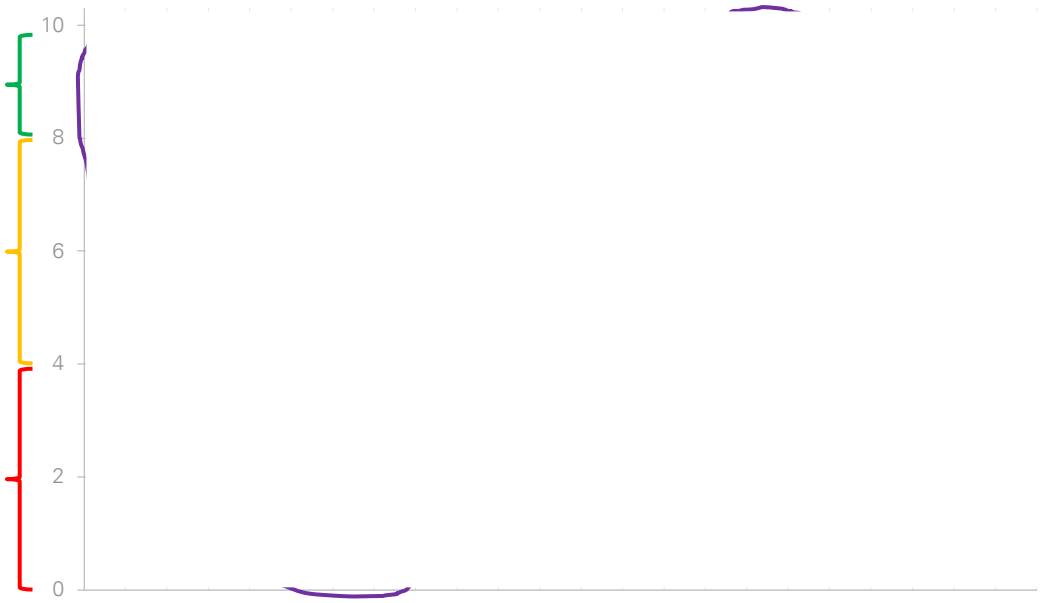


84



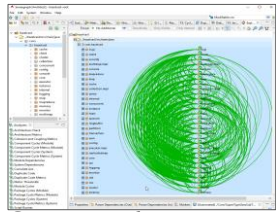


MODULARITY MATURITY INDEX (MMI)

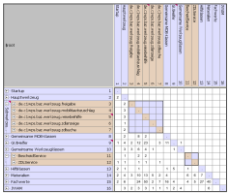
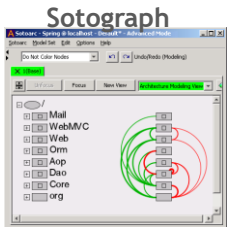
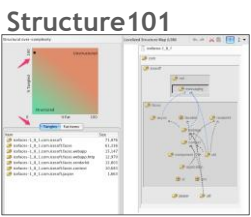


@caiolali

ANALYSING MODULARITY WITH TOOLS



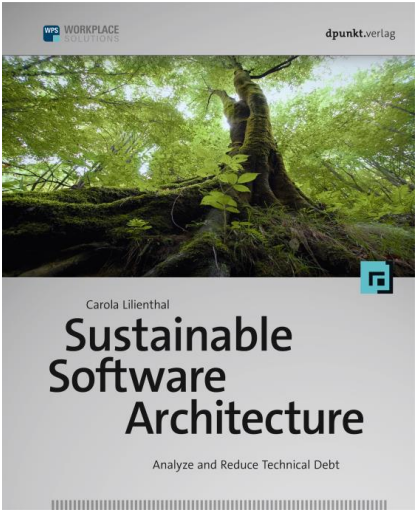
Sonargraph



Lattix

@caiolali

THANK YOU FOR YOUR ATTENTION!



@cairolali
cl@wps.de

www.sustainable-software-architecture.com

@cairolali