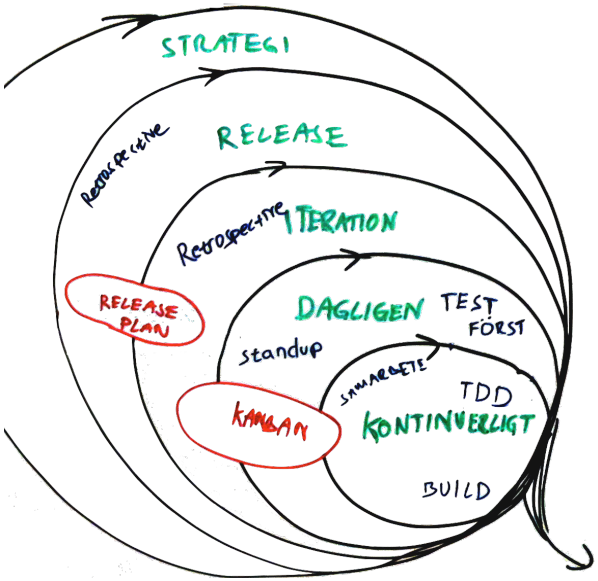
Arkitektur

- Done
  - Den Röda Tråden
  - Vad är arkitektur?
  - Vad har vi arkitekturmodellen till?
  - Hur redovisar vi en arkitektur?
  - Hur tar vi fram en arkitektur?
  - Uppgift

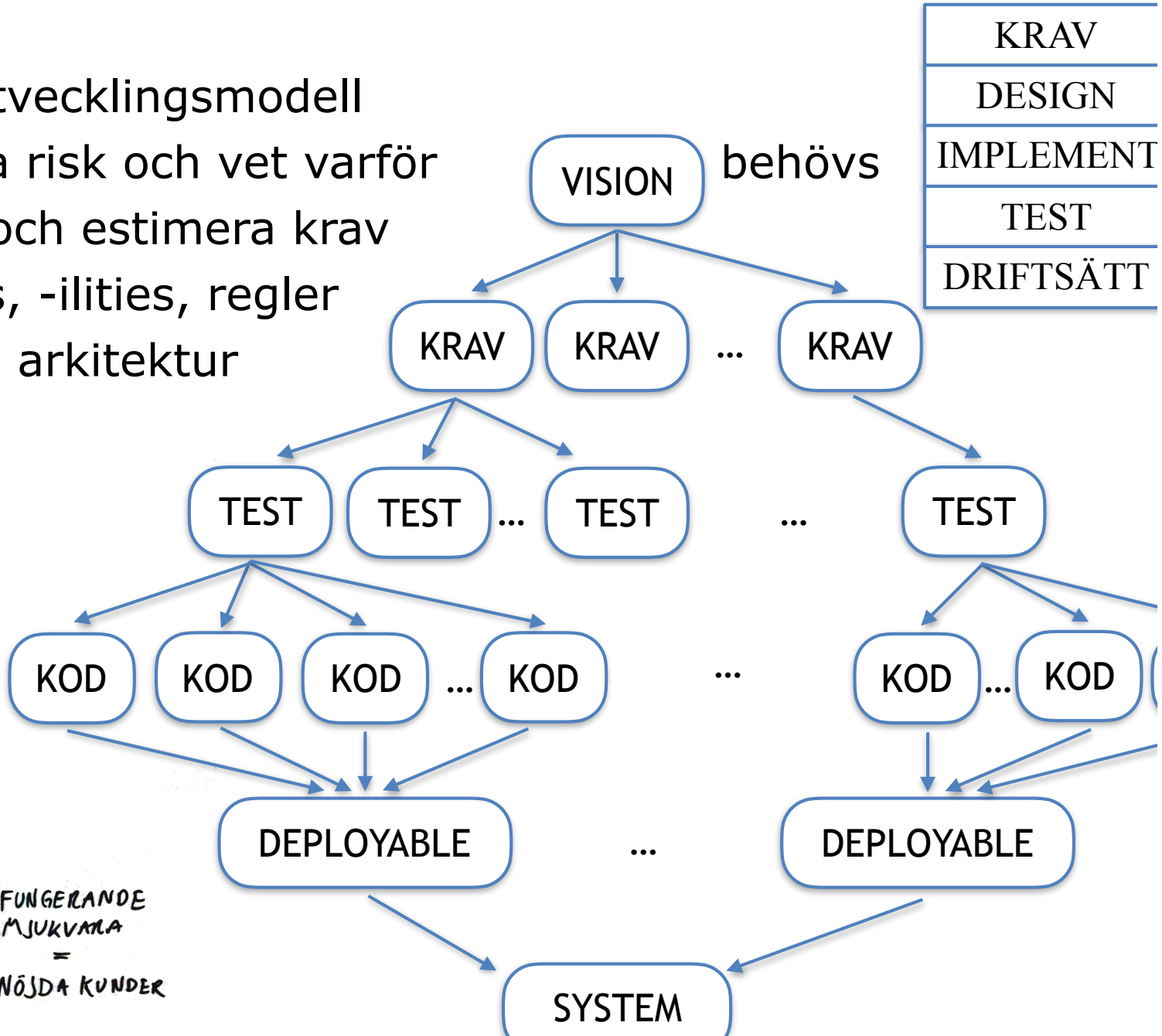
- Nu
  - Redovisning/Diskussion uppgift
  - Den Röda Tråden
  - Spotify Engineering Culture - arkitektur?
  - När sker arkitektur i RUP, XP?
  - Begreppsmodell och bounded context
  - Architectural Patterns
  - Design (om vi hinner)
  - Design Patterns (om vi hinner)

Den Röda Tråden

- Vi kan välja utvecklingsmodell
- Vi kan hantera risk och vet varför
- Vi kan skriva och estimera krav
  - User stories, -ilities, regler
- Vi kan ta fram arkitektur



FUNGERANDE MJUKVARA = NÖJDA KUNDER

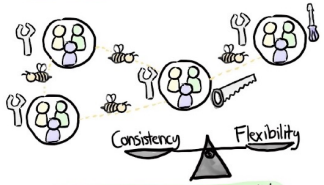


Spotify Engineering Culture

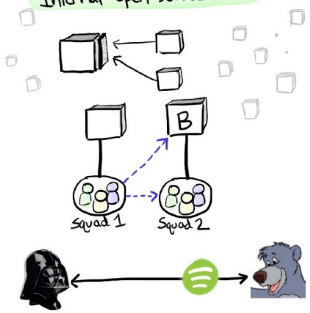
Part 1 of 2

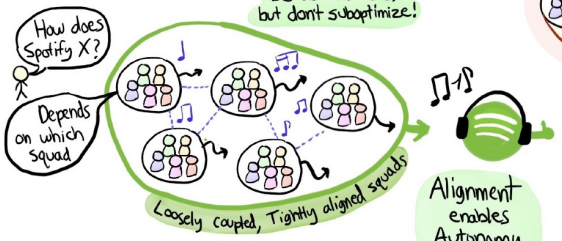
Henrik Kniberg
Jan 2014

Cross-pollination > Standardization

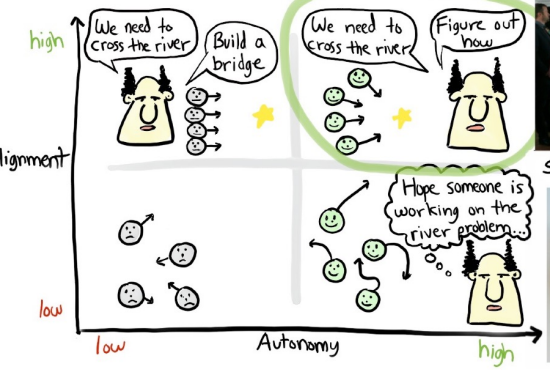


Internal Open-source model

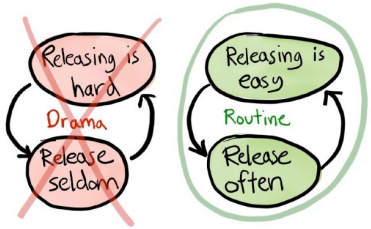




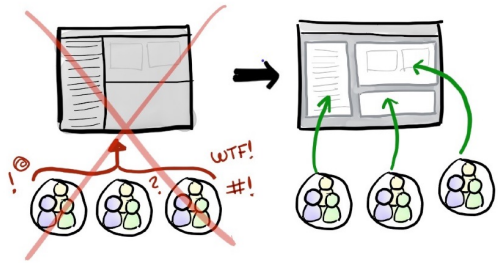
Alignment enables Autonomy



Small & frequent releases

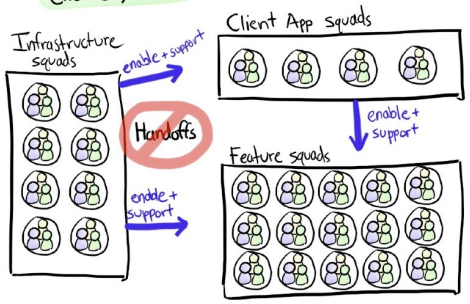


Decoupled releases

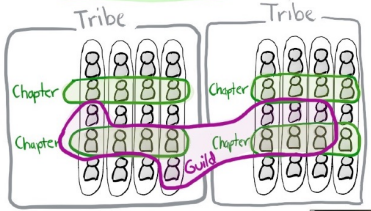


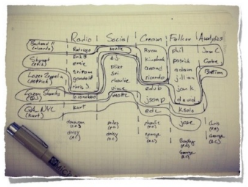
Self-service model

enable > serve



Community > Structure





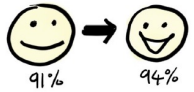
If you need to know exactly who is making decisions, you are in the wrong place.

Focus on Motivation

Hi everyone,
Our employee satisfaction survey says **91% enjoy working here,** and 4% don't.

This is of course not satisfactory, and we **want to fix it.**

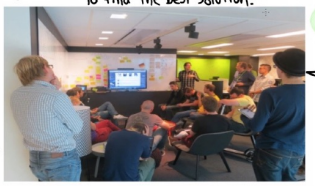
If you're one of those unhappy 4%, please contact us.
We're here for your sake, and nothing else.



Trust > Control

Agile at scale requires Trust at scale

Leader's job: Communicate what problem needs to be solved. And Why.

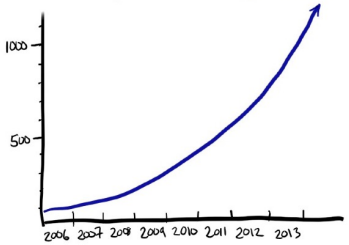


Squads' job: Collaborate with each other to find the best solution.

People > *

My colleagues are awesome!

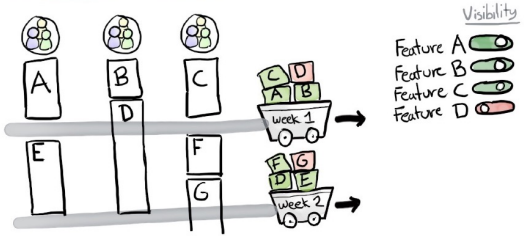
Ego



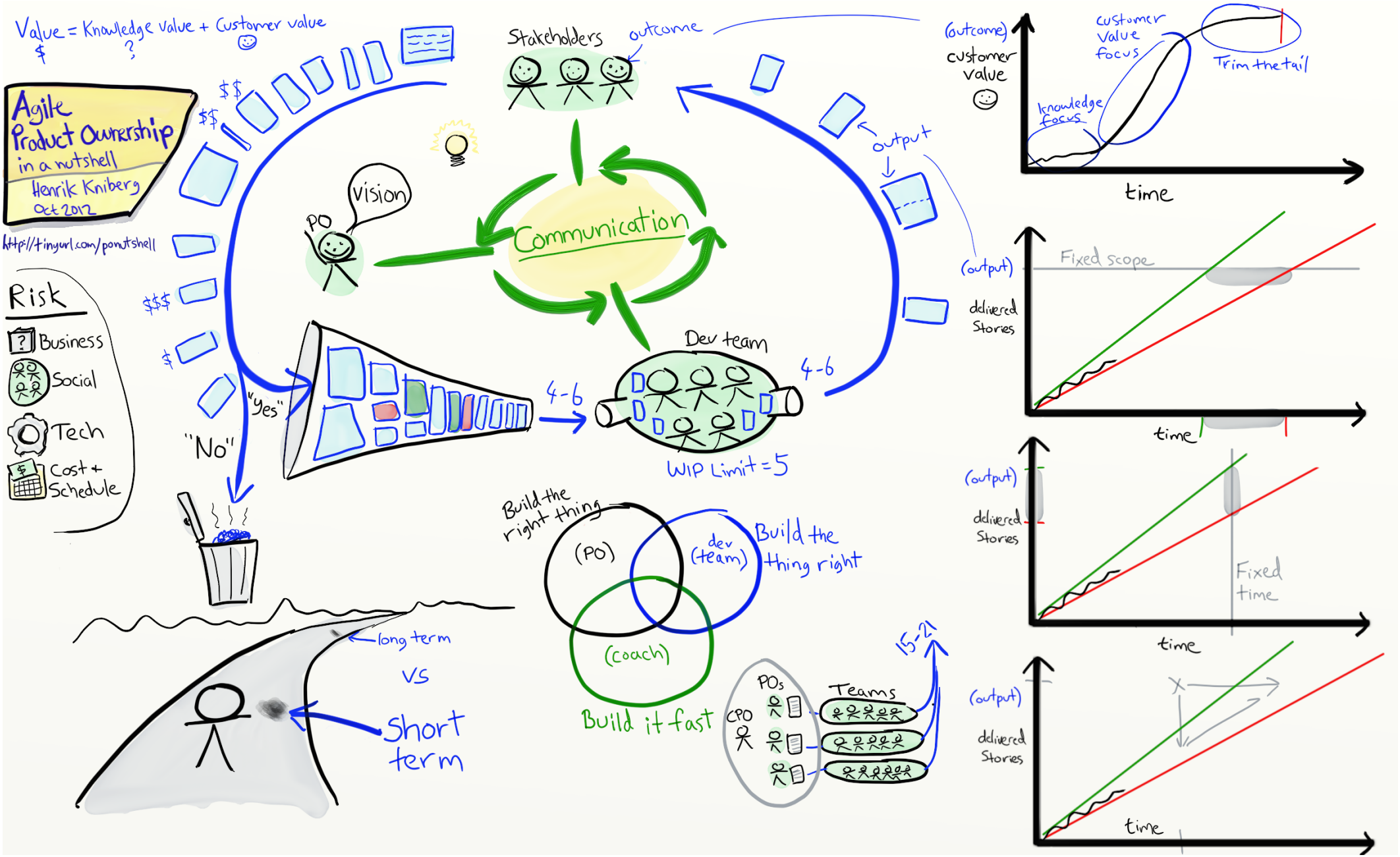
Politics

Fear

Release Trains & Feature Toggles



Varför Arkitektur?

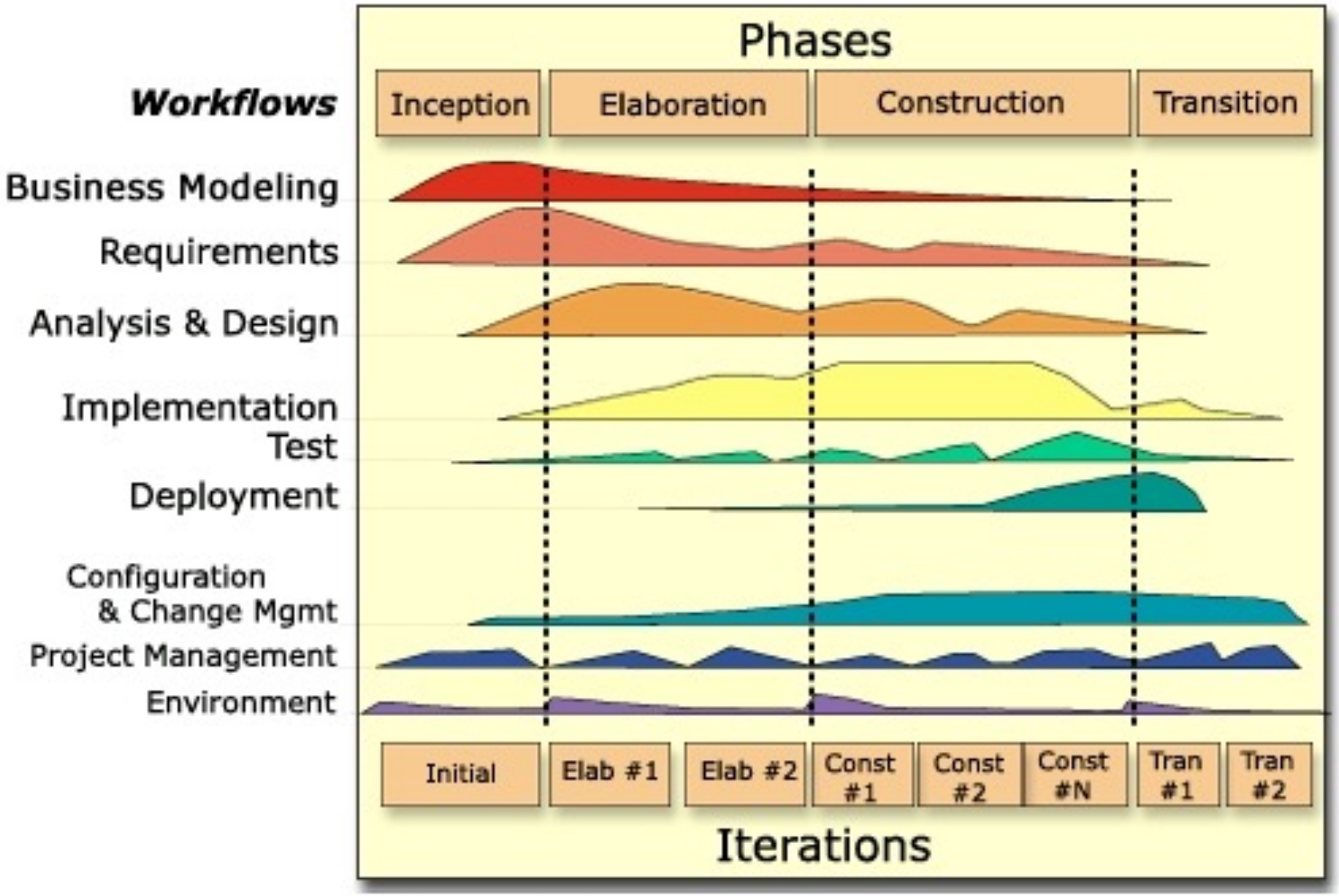


http://blog.crisp.se/2012/10/25/henrikkniberg/agile-product-ownership-in-a-nutshell

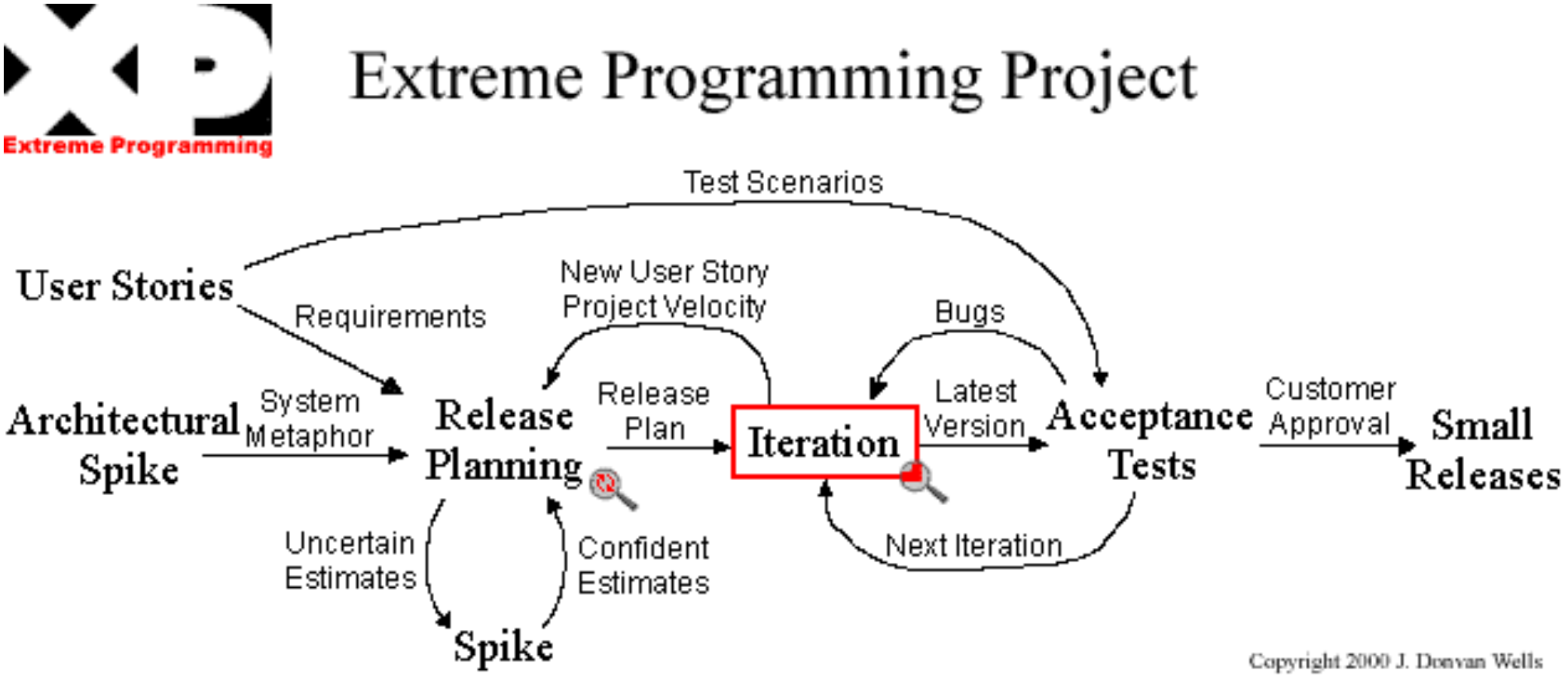
What is software architecture

https://www.youtube.com/watch?v=Rn1g6V-vlHw

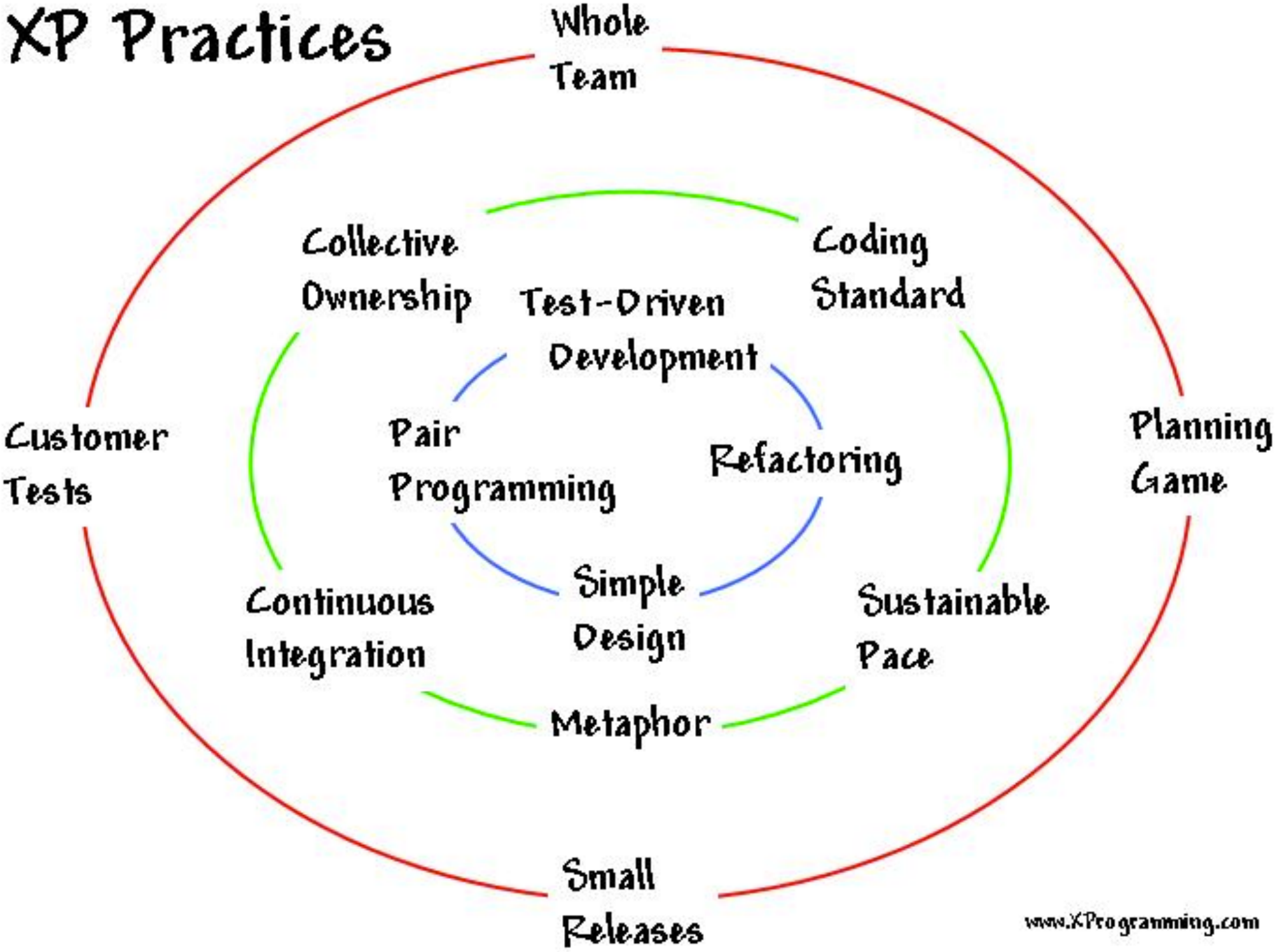
Arkitektur när?



Arkitektur när?

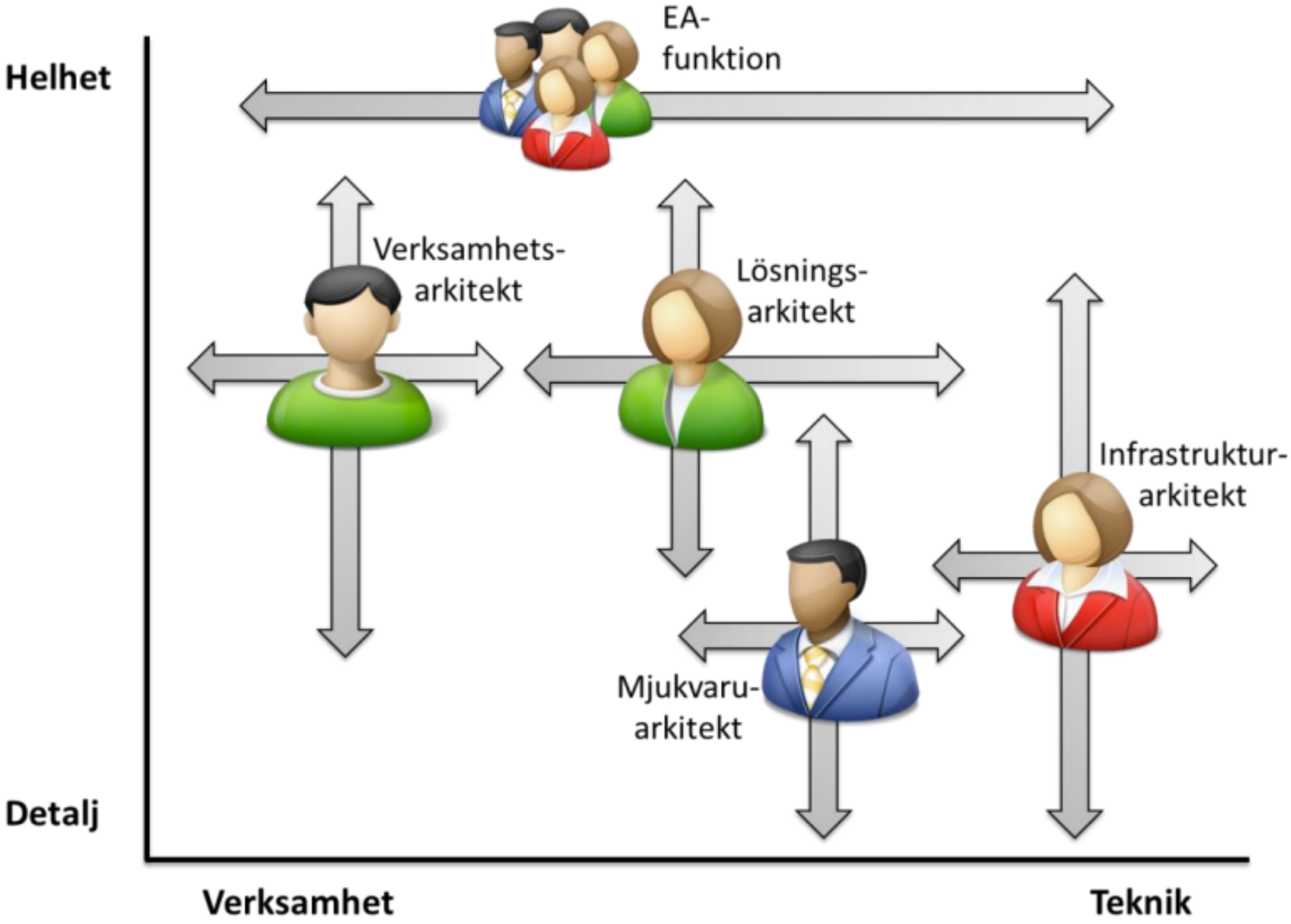


http://www.extremeprogramming.org/map/project.html

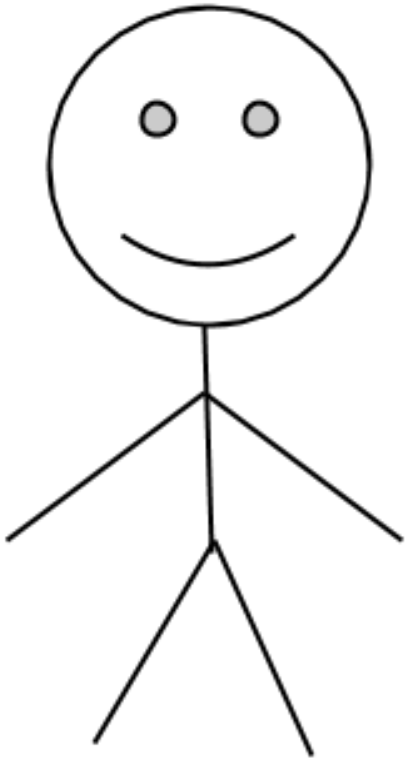

XP Practices
Whole Team
Collective Ownership
Coding Standard
Test-Driven Development
Pair Programming
Refactoring
Customer Tests
Planning Game
Continuous Integration
Simple Design
Sustainable Pace
Metaphor
Small Releases
www.XProgramming.com

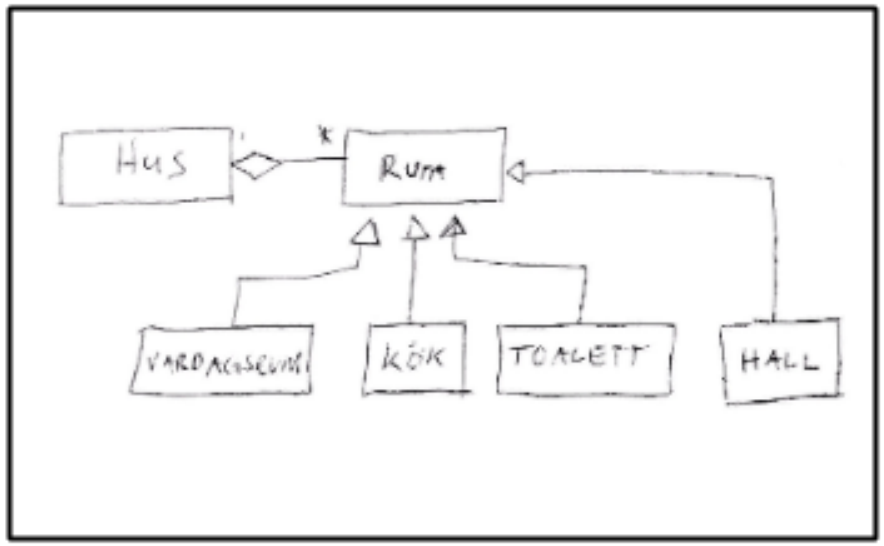
http://www.iasa.se/wp-content/uploads/2012/05/IASA-Arkitektroller-2012.pdf

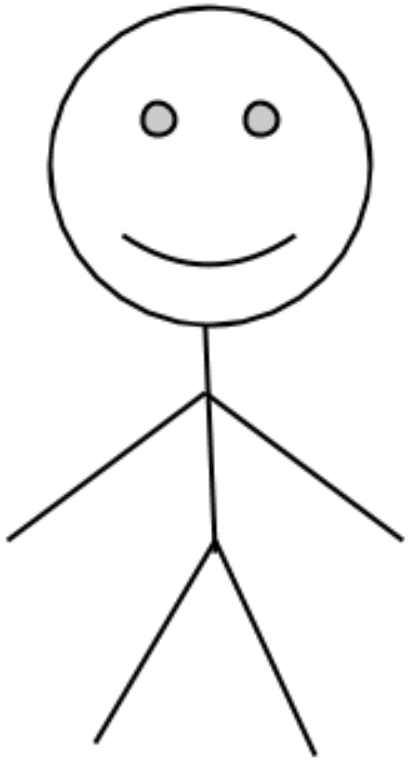
Arkitektroller IASA



Begreppsmodell







Varför begreppsmodell?

- Ökad produktivitet!
  - Effektivare kommunikation
  - Tydlighet, undvika missförstånd
  - Minskad kognitiv last
  - Hjälper oss förstå domänen

Bord för fyra

Bord för fyra

Bounded context

- Gränsen där ett begrepp blir luddigt - program är inte bra på ludd!
- Ofta går gränsen vid teamet —> En begreppsmodell per team kan behövas!

Arkitekturmönster

- General structure; e.g., shared repository, **layers**, pipes and filters, **monolith**
- Distributed systems; e.g., client-server, **REST, SOA, Microservices**
- Interactive systems; e.g., Model-View-Controller (MVC), MVVM
- Integration: ESB
- Data Architecture: Data Warehouse
- Infrastructure: Load Balancer

Monolith

- Applikation som byggs ihop till en enda stor komponent
- Monolitens delar är fortfarande komponenter
- Utvecklas, testas, integreras och deployas som en enhet.

Monolith

- Fördelar
  - Enkelt se hur allt hänger ihop
- Nackdelar
  - Hög coupling
  - Svårt dela upp ansvar över flera team
  - Kan inte testa, deploya separata delar - långa cykler
  - Svårt resonera om när den blir för stor.

SOA

Service **O**riented **A**rchitecture

- Modellera arkitektur som tjänster

- Varje tjänst är en autonom implementation av funktionalitet

- Messaging eller RPC

- Interface baserat på standardiserade protokoll som XML, SOAP, WSDL, UDDI

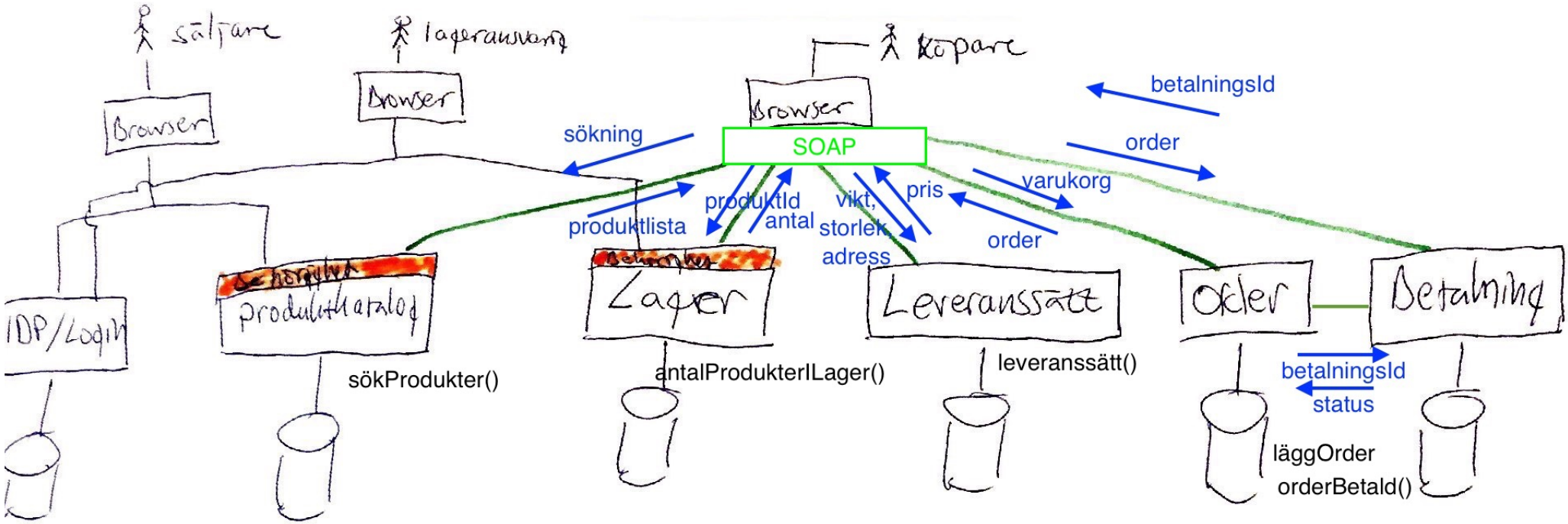
- Tänkt att stödja runtime-beslut om vilken endpoint som ska användas - Service Lookup.

- Ofta implementerat med en ESB

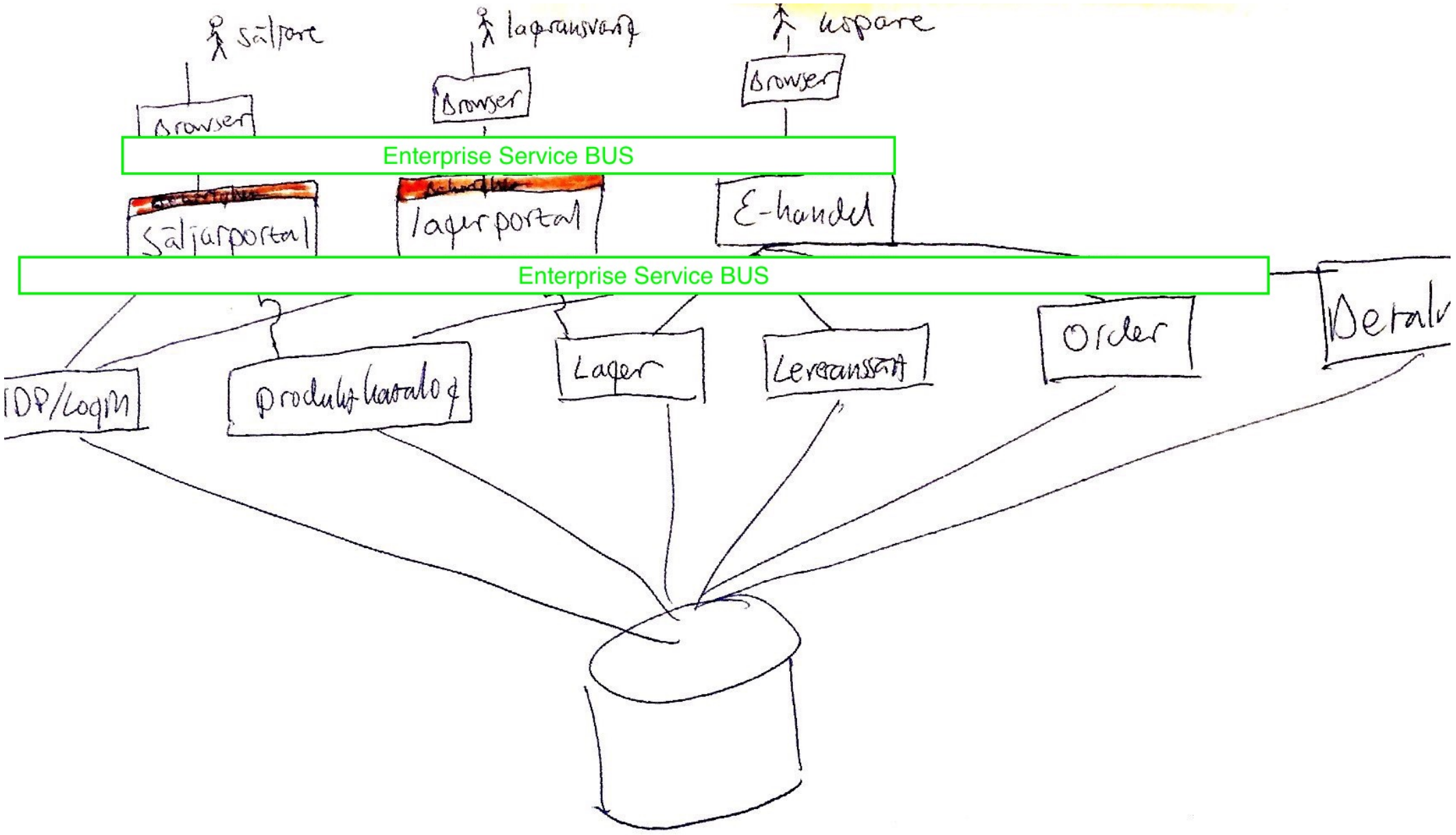
SOA

- Fördelar
  - Standardiserat
  - Enkelt generera klient
- Nackdelar
  - Tungjobbat och omständigt med SOAP, WSDL, UDDI, …
  - I praktiken kan vara svårt att få klient att funka ändå
  - Ju större specifikare tjänst, desto svårare att återanvända
- Användning
  - e2e

Exempel E-handel



Exempel E-handel



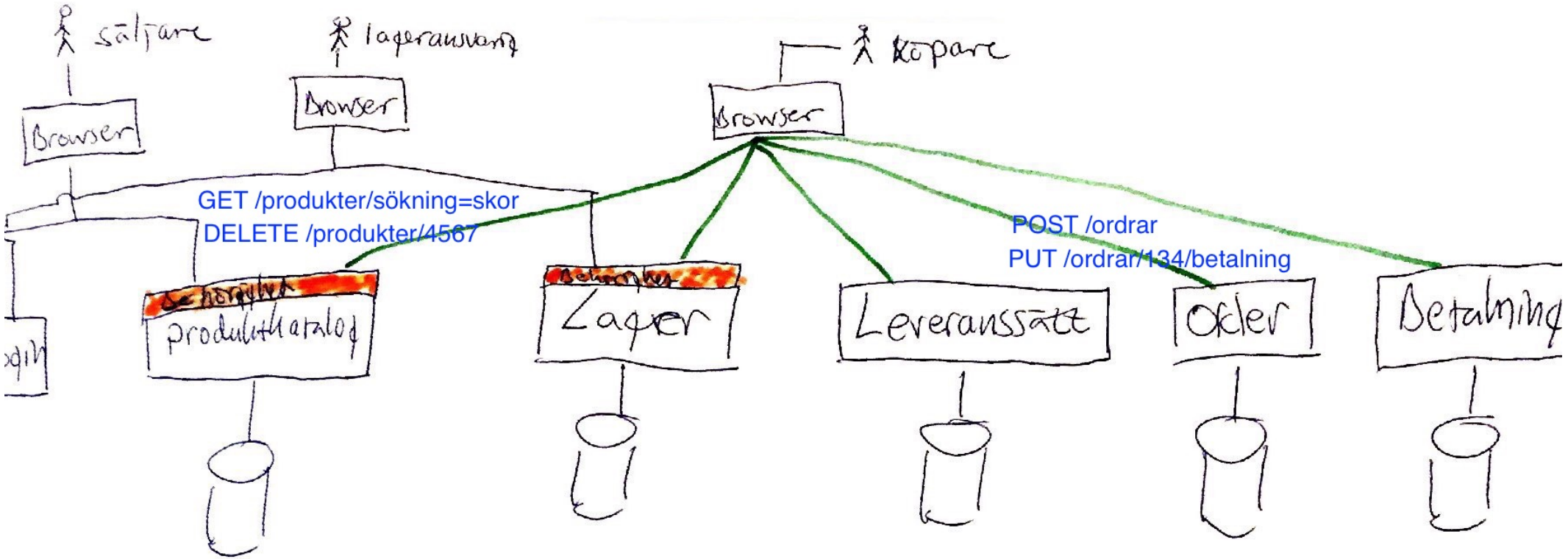
Microservices

- Modellera arkitektur som små tjänster
- Varje tjänst är en autonom implementation av funktionalitet
- Autonoma tjänster - low coupling, high cohesion
- Enkla protokoll som https, ftp, etc.
- Ingen ESB!
- “SOA reincarnated”

Microservices

- Fördelar
  - Lättviktigt
  - Autonoma tjänster som kan deployas separat
- Nackdelar
  - Många tjänster —> ökad komplexitet!
  - Transaktioner över flera tjänster - don’t!

Exempel E-handel



REST

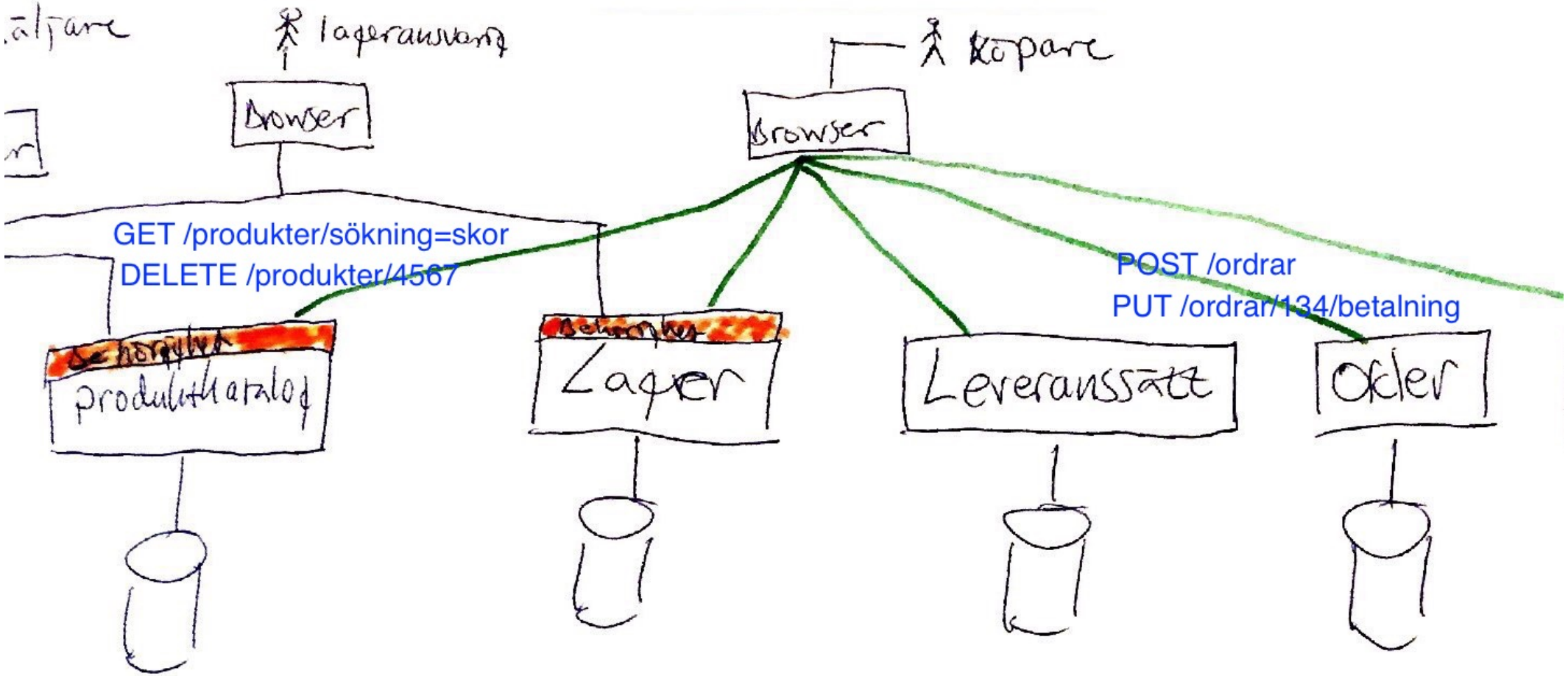
Representational **S**tate **T**ransfer

- Modellera arkitektur som resurser
- Varje resurs används med HTTP-verben GET, POST, PUT, DELETE, etc…
- Använder befintlig HTTP-teknik för skalning
- Ej standardiserat interface
- Payload oftast XML eller JSON.

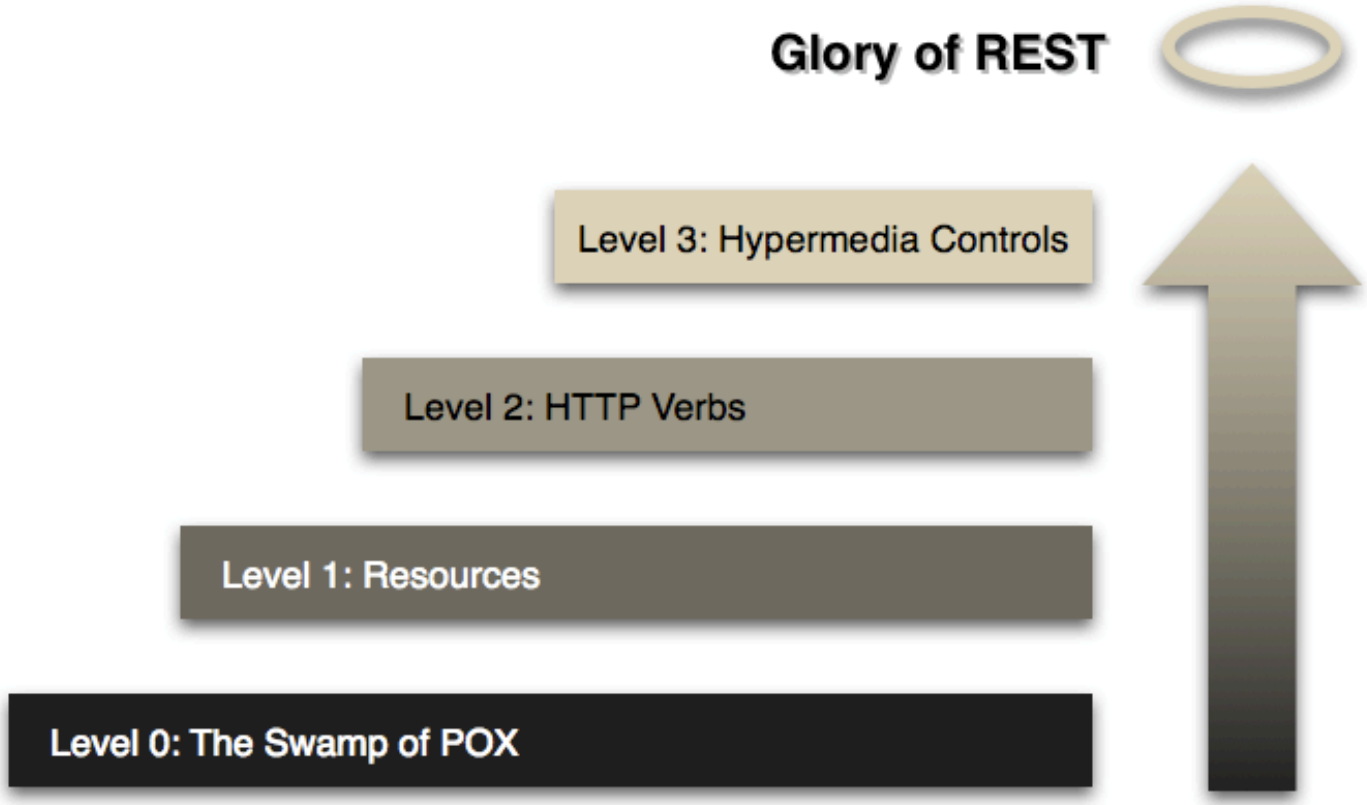
REST

- Fördelar
  - Standardiserat
  - Lättviktigt
- Nackdelar
  - Payload ofta odefinierad - få på nivå 3!
  - HATEOAS vs Swagger
- Användning
  - Frontend-backend
  - API:er
  - e2e
  - …

Exempel E-handel

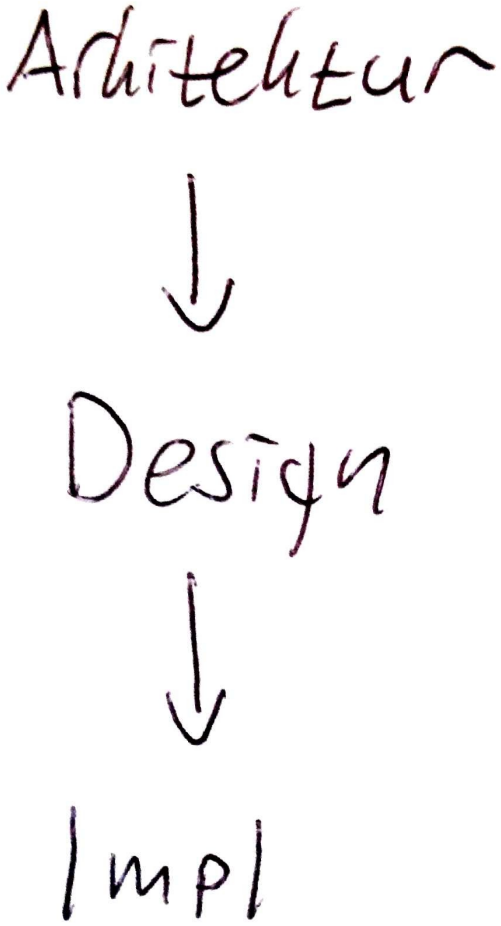


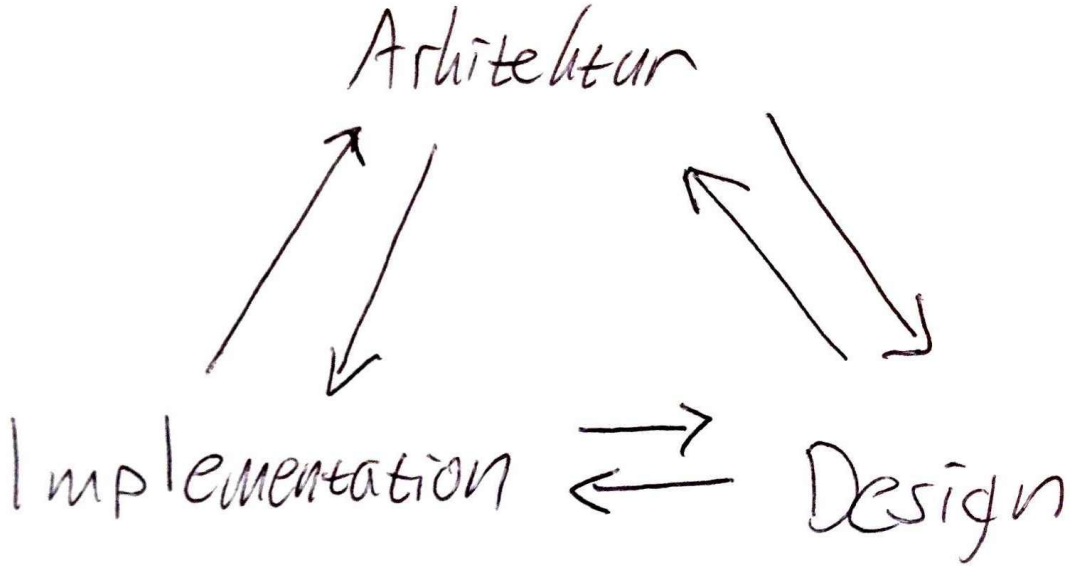
Richardson Maturity Model



https://martinfowler.com/articles/richardsonMaturityModel.html

Vilken föredrar vi?





Design!

Till nästa vecka

- Kolla in filmen, 10:45

The Software Craftsmanship Imperative:

https://www.youtube.com/watch?v=fdeLcImL7e4