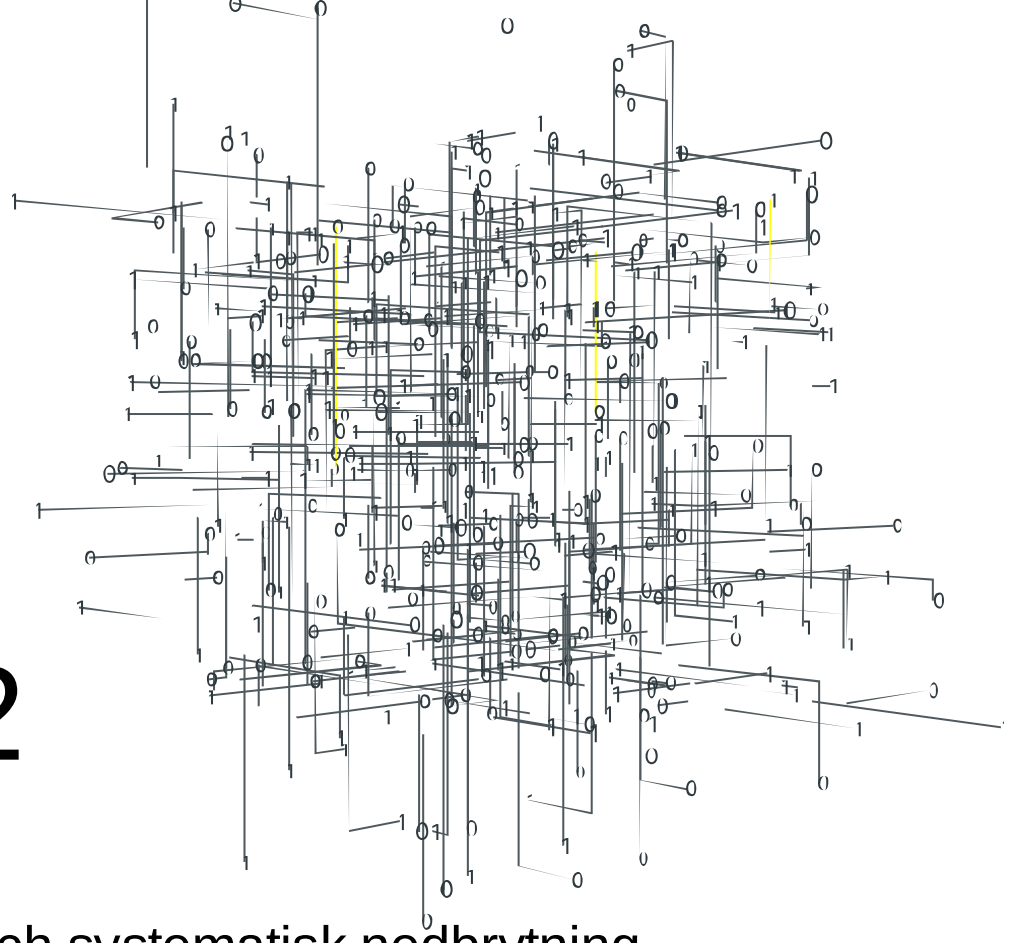


2DT902

Design, modeller och systematisk nedbrytning

Linnæus University

Dagens föreläsning

Mjukvarudesign

Designprinciper

Designkvalitet

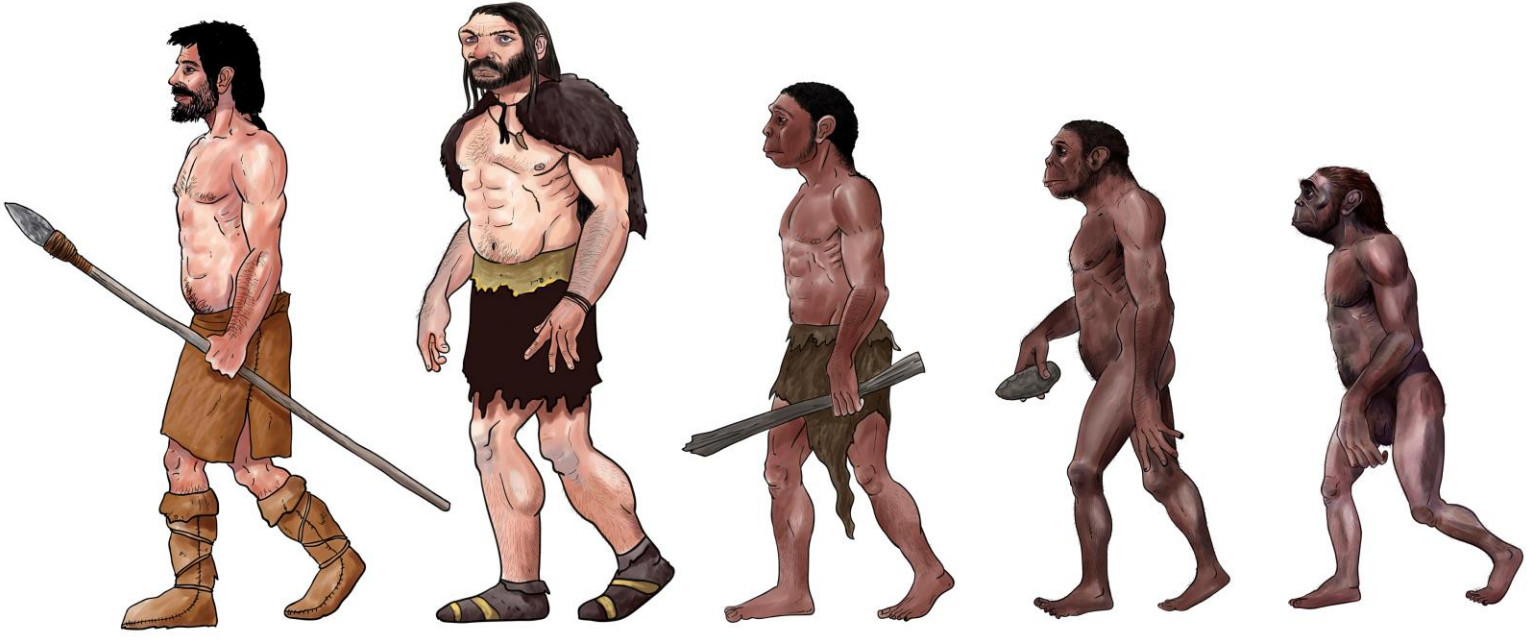
Problemlösning

Designkunskap

‘The fundamental problem is that designers are obliged to use current information to predict a future state that will not come about unless their predictions are correct. The final outcome of designing has to be assumed before the means of achieving it can be explored: the designers have to work backwards in time from an assumed effect upon the world to the beginning of a chain of events that will bring the effect about.’

J. Christopher Jones, Design Methods: Seeds of Human Futures (Jones, 1970).

Vad gör vi?



Människor designar, skapar och använder verktyg!

Men, vad är det då vi gör?

Polya's problemlösningsprocess

1. Förstå problemet
2. Ta fram en plan för att lösa problemet
3. Genomför planen
4. Utvärdera resultatet

Mjukvarudesign är problemlösning i mjukvara

Lösningen är mjukvara

Vi använder mjukvaruspecifika metoder

Hantera ”wicked problems”

Lösningar kan inte testas direkt

Det finns flera förklaringar av ett problem

Lösningar är bättre eller sämre

Ingen klar definition

Inga stoppregler

Inga alternativa lösningar

Varje problem har kopplingar till andra

Problemlösaren har inte rätt att ha fel.

Problemlösningen är en unik situation

tillhandahålla API:er och datamodeller för infrastrukturoperatörer, tjänsteleverantörer och enhetsleverantörer

tillhandahålla en plattform för slutanvändare som stöder sömlös enhetsintegration, regelbaserad automatisering och ett konfigurerbart användargränssnitt

Inga stoppregler?

tillhandahålla en plattform för slutanvändare som stöder sömlös enhetsintegration, regelbaserad automatisering och ett konfigurerbart användargränssnitt

“wicked problems”

det saknas kriterier som kan användas för att fastställa när lösningen på ett problem har hittats, så att ytterligare arbete inte kommer att kunna förbättra det.

Lösa “wicked problems”

tillhandahålla API:er och datamodeller för infrastrukturoperatörer, tjänsteleverantörer och enhetsleverantörer

tillhandahålla en plattform för slutanvändare som stöder sömlös enhetsintegration, regelbaserad automatisering och ett konfigurerbart användargränssnitt

Lösningar kan inte testas direkt

Det finns flera förklaringar av ett problem

Lösningar är bättre eller sämre

Ingen klar definition

Inga stoppregler

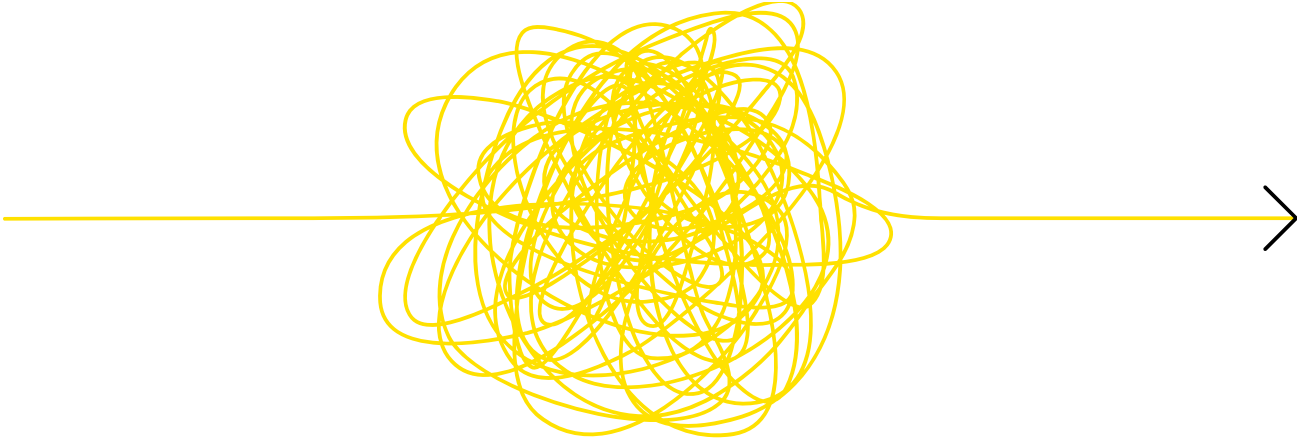
Inga alternativa lösningar

Varje problem har kopplingar till andra

Problemlösaren har inte rätt att ha fel.

Problemlösningen är en unik situation

Bidrar till problemets komplexitet



Hur hanterar vi komplexitet?

Hur hanterar vi komplexitet?

Abstraktion

- Förenkling
- Beskriv delarna endast med relevanta egenskaper (beror på kontext).
- Exempel, dataabstraktion, instruktionsabstraktioner

Nedbrytning (Decomposition)

- *Problemet delas upp i delproblem som i sin tur delas upp i delproblem som i sin tur…*
- Problemet delar in lösningen i delar
- Söndra och härska (divide n' conquer)
- Minimera beroenden

Hur hanterar vi komplexitet?

Från svarta till genomskinliga lådor

- Reducera mängden information
- Öppna inte upp för tidigt – ”maskburkar”
- Iterativt och stegvis-förfining

Inkrementellt

- Arbeta på delar av problemet
- Reducerar problemstorleken

Stegvis förfining

Den första generella designmetoden för mjukvara.

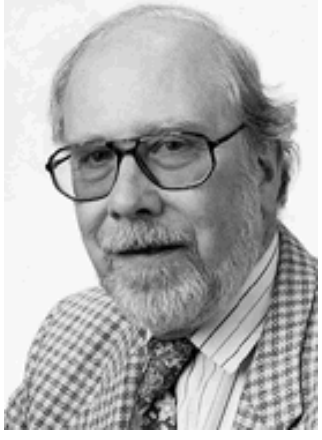
Program Development by Stepwise Refinement

Niklaus Wirth
Eidgenössische Technische Hochschule
Zürich, Switzerland

these two purposes in mind. Some well-known techniques are briefly demonstrated and motivated (strategy of preselection, stepwise construction of trial solutions, introduction of auxiliary data, recursion), and the program is gradually developed in a sequence of *refinement steps*.

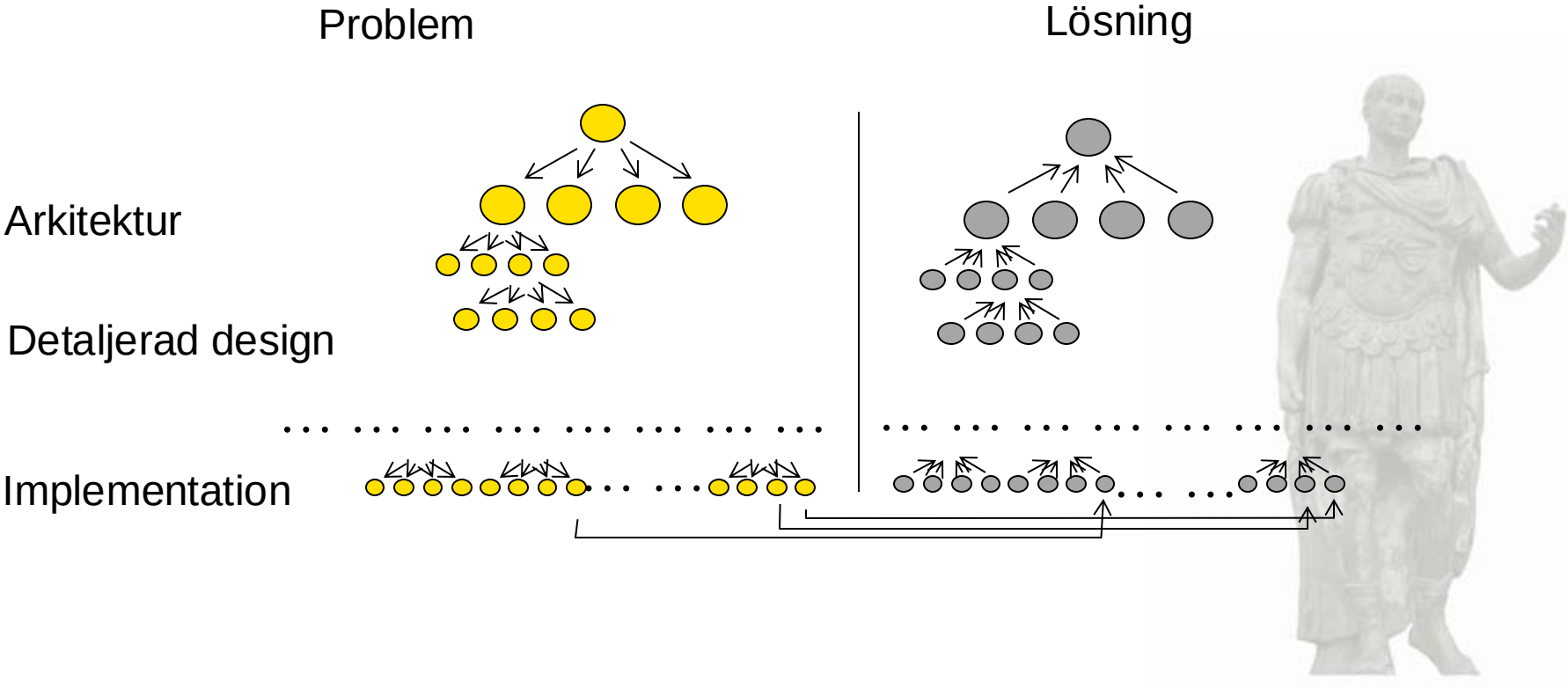
In each step, one or several instructions of the given program are decomposed into more detailed instructions. This successive decomposition or refinement of specifications terminates when all instructions are expressed in terms of an underlying computer or programming language, and must therefore be guided by the

- Programkonstruktionen sker i ett antal förfiningssteg.
- I varje steg delas en given uppgift upp i ett antal deluppgifter.
- Varje förfining i beskrivningen av en uppgift kan följas av en förfining av beskrivningen av den data som kopplar samman deluppgifter.
- Förfining av beskrivningen av program- och datastrukturer kan och bör ske parallellt.



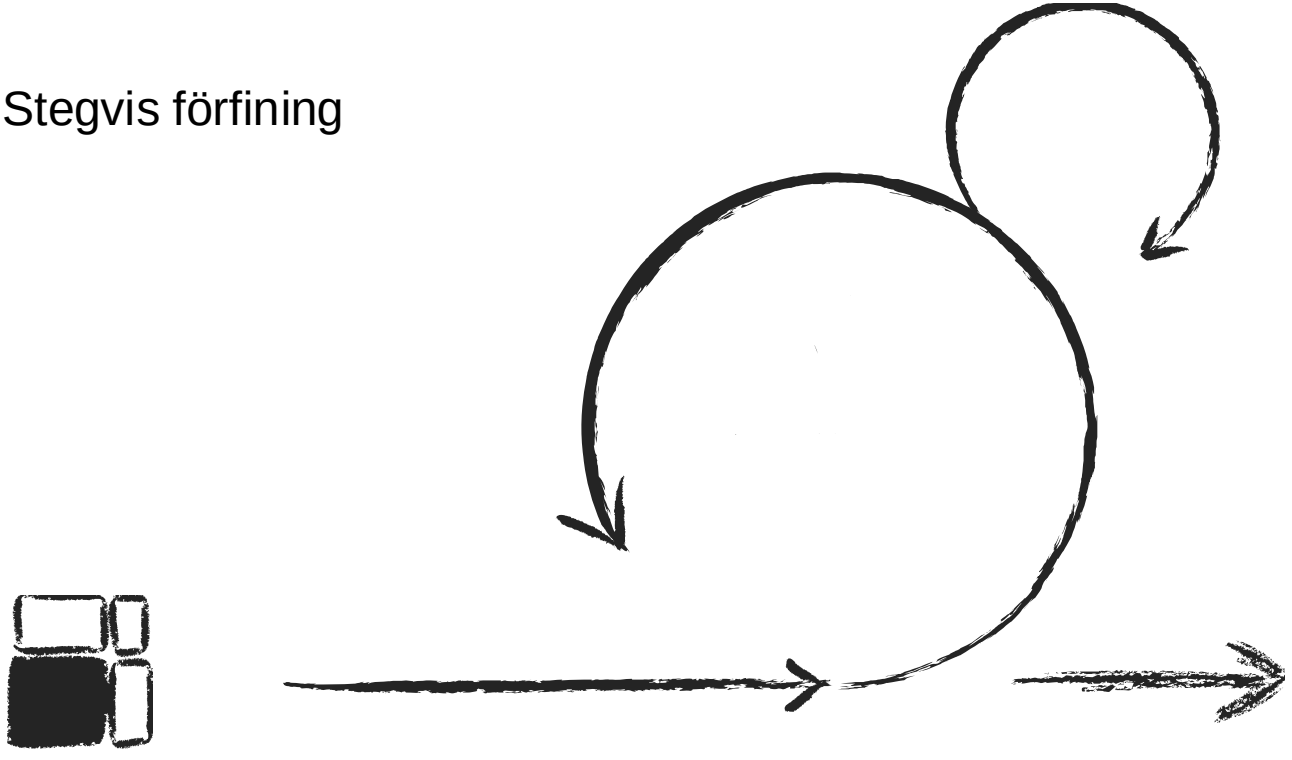
Niklaus Wirth

Divide n’ Conquer



Iterativ utveckling

Stegvis förfining



Små uppgifter– Små kontroller – Små förändringar

Inkrementell utveckling

Arkitekturmodeller

Motivation

Modellens roller

Modeller stödjer kommunikation

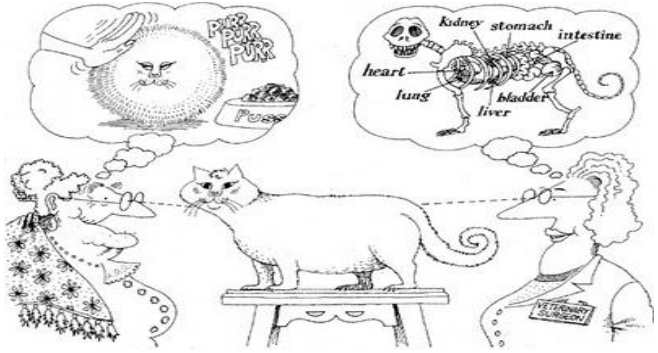
De utgör ett gemensamt språk med ett ordförråd och semantik

Modeller konstueras för att möjliggöra resonemang inom ett idealiserat logiskt ramverk.

Idealiserad innebär att modellen kan göra explicita antaganden som är kända för att vara falska i någon detalj → förenklingar!

Modeller ger stöd för att hantera komplexitet. Modellernas uppbyggnad och koncept ger stöd.

Abstraktion



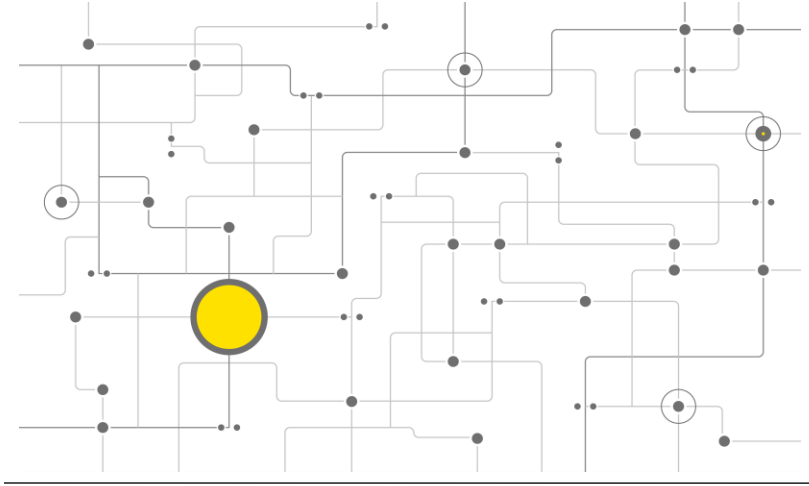
"en förenklad beskrivning, eller specifikation, av ett system som betonar vissa av systemets detaljer eller egenskaper samtidigt som andra undertrycks.

En bra abstraktion är den som betonar detaljer som är viktiga för läsaren eller användaren och undertrycker detaljer som - åtminstone för tillfället - är ointressanta." – *Shaw, Mary. 1984*

Modularitet

Dela upp ett system i delar

Modularitet i mjukvaruutveckling är ett mått på hur uppdelat ett system är i delar (moduler eller komponenter). Varje del har ett specifikt ansvar (funktionalitet) och fungerar självständigt i så hög grad som möjligt.



Hierarki

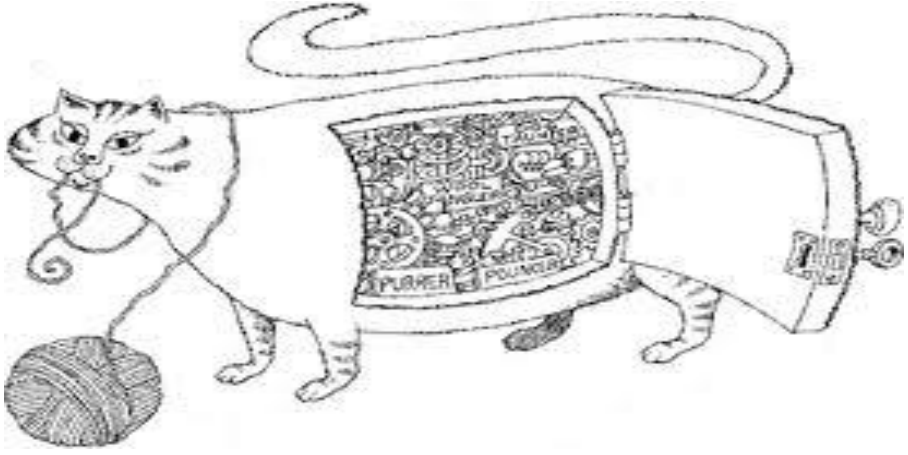
Bygg hierarkier – Sätt samman delsystems till större system

Hierarki är (rang)ordningen av abstraktioner

Helhet – Delar

Generalisering – Specialisering

Inkapsling och information hiding



"processen att skilja ut elementen i en abstraktion som utgör abstraktionens struktur och beteende; inkapsling tjänar till att separera det gränssnittet som utgör en abstraktions kontrakt och den faktiska implementationen."

”kontrollera direkt tillgång till struktur och beteende så att endast det som finns i det öppna kontraktet blir åtkomligt från andra abraktioner"

Designprinciper

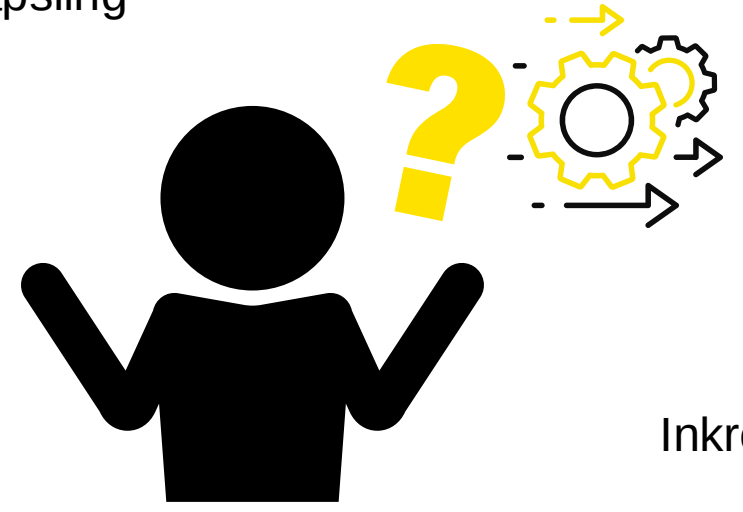
Från svarta lådor till genomskinliga för att lösa wicked problems

Sätt ihop allt

Dölj information

Inkapsling

Abstraktion



Hierarkier

Modularitet

Iterativt

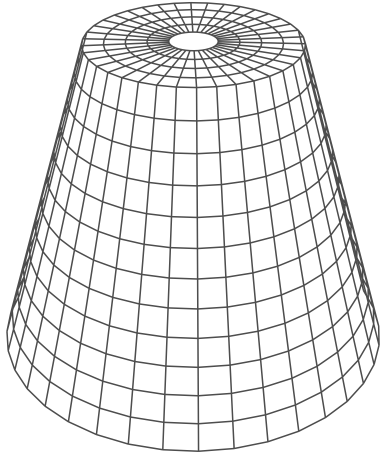
Inkrementellt

Partitionering – Nedbrytning

Modularitet

Från svarta till genomskinliga lådor

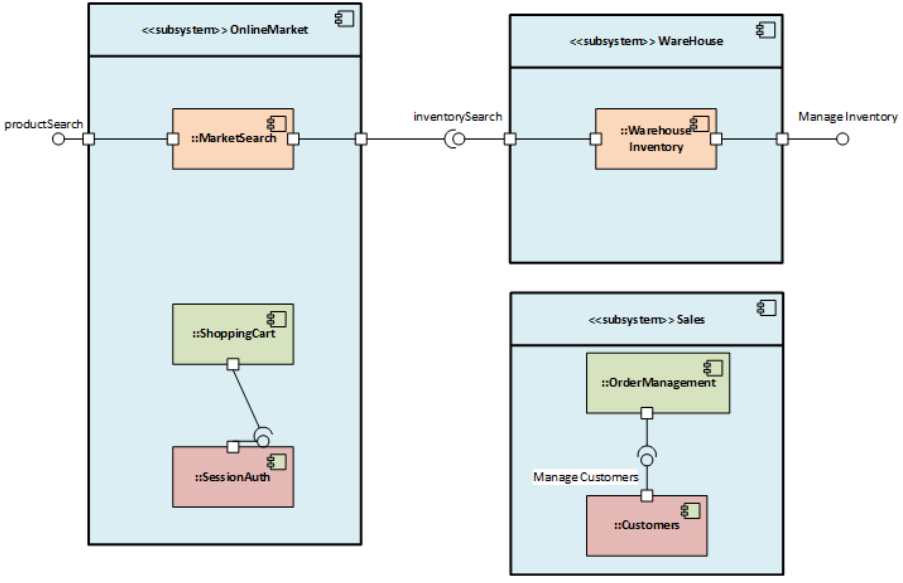
1. Dela upp problemet
2. Delar med ansvar
3. Divide n’ conquer



Perspektiv

1. Lager eller distribution
2. Funktionalitet – Gränssnitt
3. Separation av områden (concerns)

Ansvar



Fördela Ansvar på Abstraktionerna

- Funktionalitet – Gränssnitt
- Tjänster

Från svarta till genomskinliga lådor

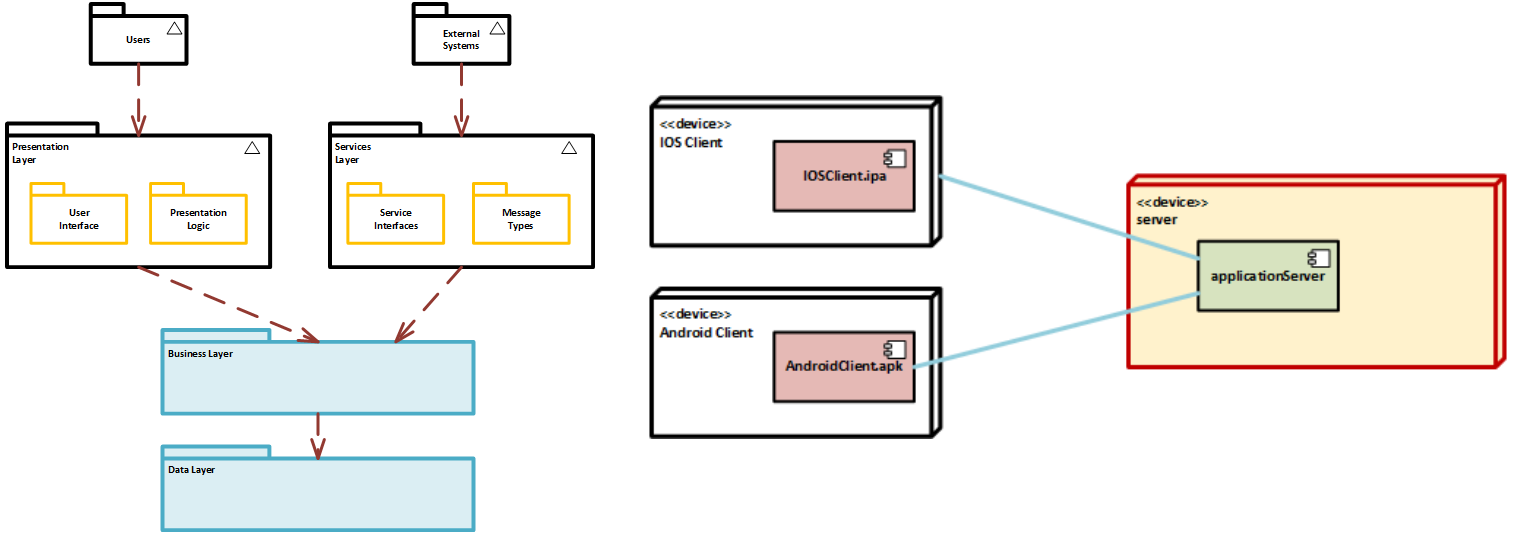
– Börja på den högsta nivån

– Förfina uppdelning och ansvar iterativt!

Perspektiv

Lager & distribution

- Lager – döljer komplexitet (logiska modeller)
- Distribution – dela upp beräkningarna (fysisk)
- Allokera ansvar



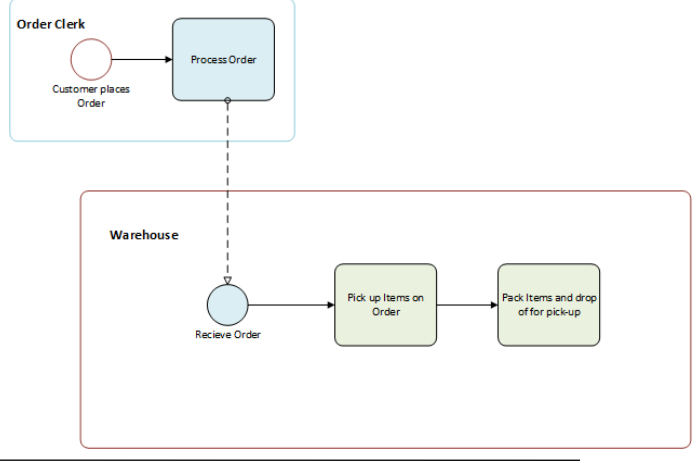
Perspektiv

Affärsprocess

Ansvar

Processer

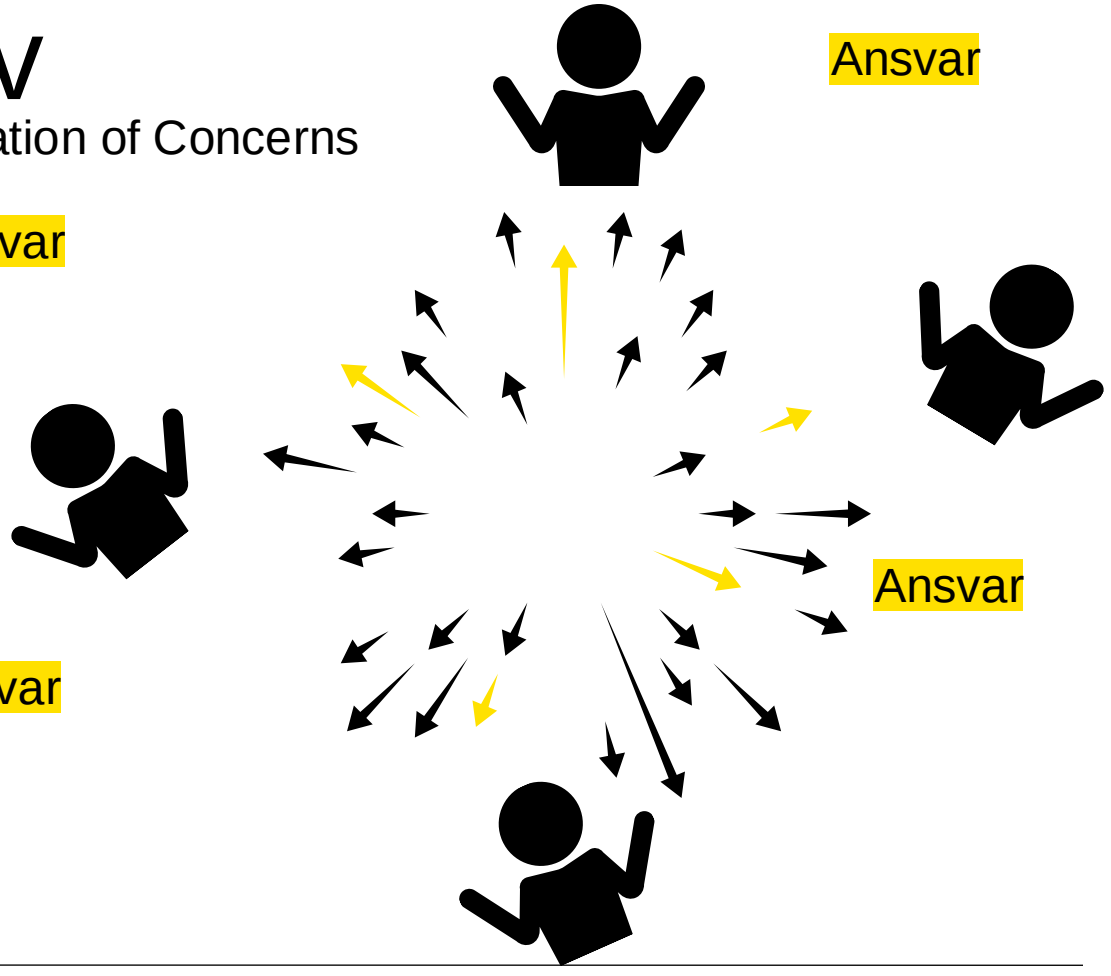
I. Kunden beställer via telefon med kreditkort
II. Ordermottagare debiterar kreditkort
III. Kontorist antecknar SKU och kunduppgifter
IV. Kontorist mejlar SKU och detaljer till lagret
V. Varan är packad och försedd med etikett
VI. Frakt till kunden påbörjas



Perspektiv

Nyckelområde – Separation of Concerns

funktionalitet
säkerhet
prestanda
(säkerhet) safety
persistens
tillgänglighet
osv.

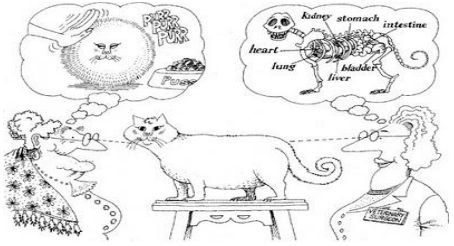


Design purpose (syfte)

Var tydlig med syftet med designen.
Varför gör du denna arkitekturdesign?

- Dela upp det funktionella ansvaret
- En abstrakt systemöversikt
- Testa en möjlig lösning
- Beskriva en del av ett system (eller hela) när utvecklingen är igång.

1:a nivån nedbrytning



Vecklar ut komplexiteten

Verktyg för förenkling

- **abstraktion**
- modularitet
- hierarki
- inkapsling

"en förenklad beskrivning, eller specifikation, av ett system som betonar vissa av systemets detaljer eller egenskaper samtidigt som andra undertrycks.

En bra abstraktion är den som betonar detaljer som är viktiga för läsaren eller användaren och undertrycker detaljer som åtminstone för tillfället är ointressanta." – *Shaw, Mary. 1984*

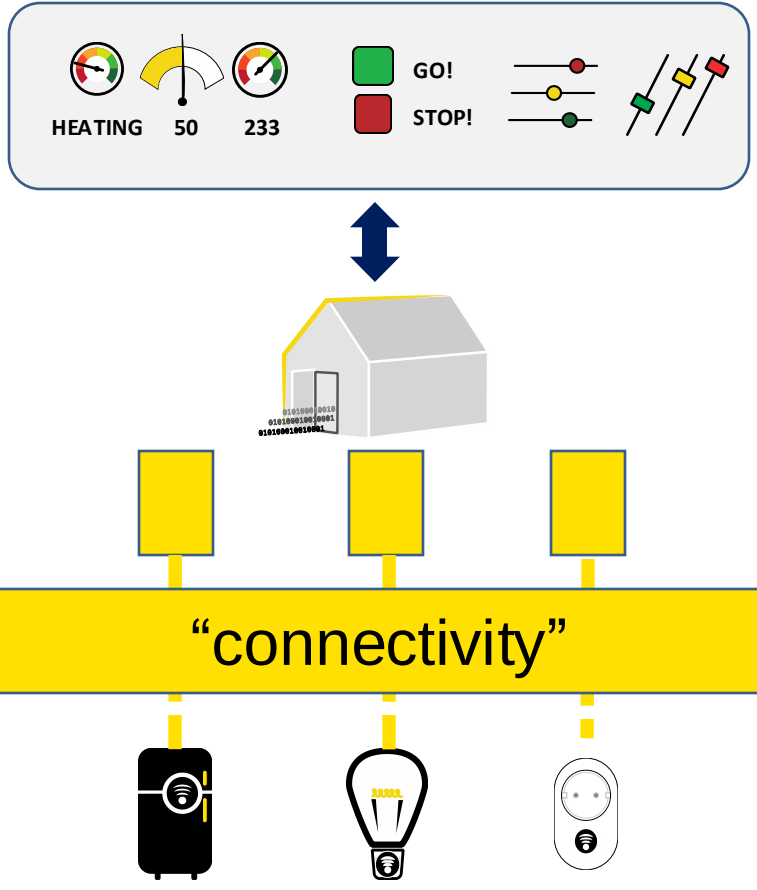
openHAB 1:a nivån abstrakt

tjänster och applikationer

kommunikation

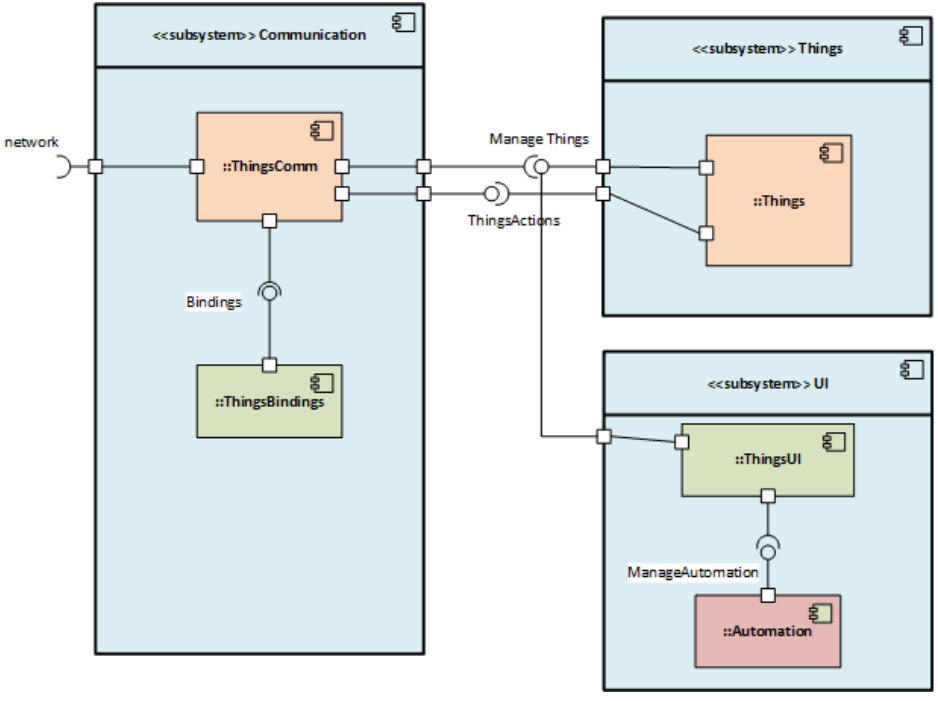
items

”things”



openHAB

1:a nivån funktionalitet

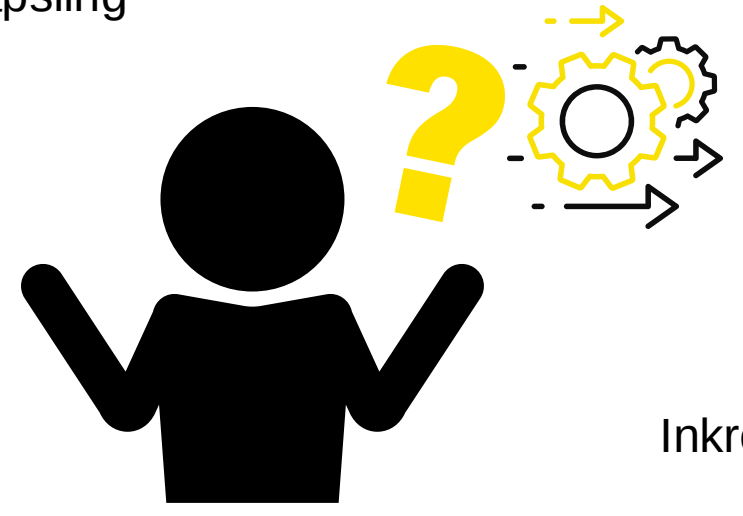


Fungerar i stort så väl som i smått

Dölj information

Inkapsling

Abstraktion



Hierarkier

Modularitet

Iterativt

Inkrementellt

Från svarta lådor till genomskinliga för att lösa wicked problems

Mjukvaruarkitektur

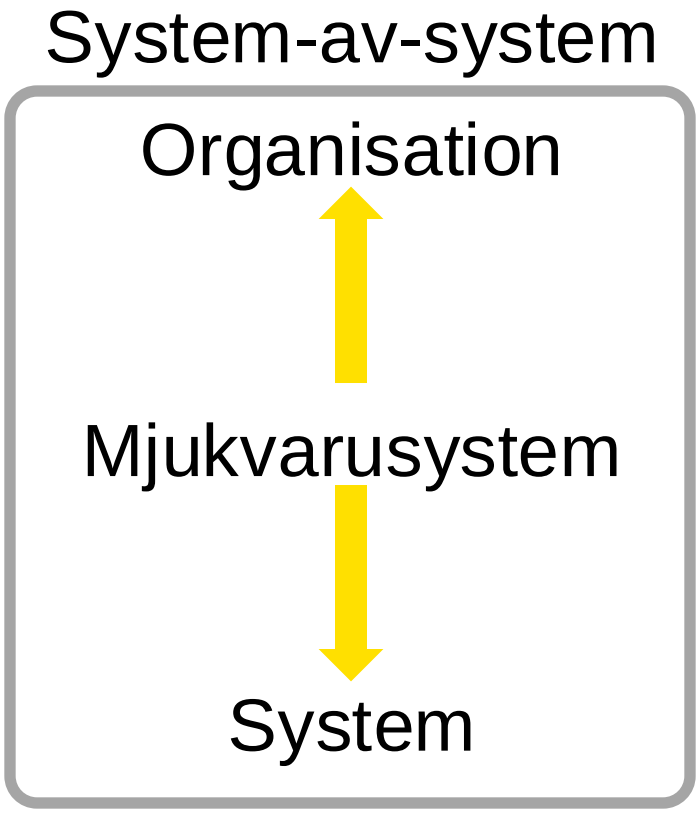
Sammanhang

Mjukvara finns i ett sammanhang

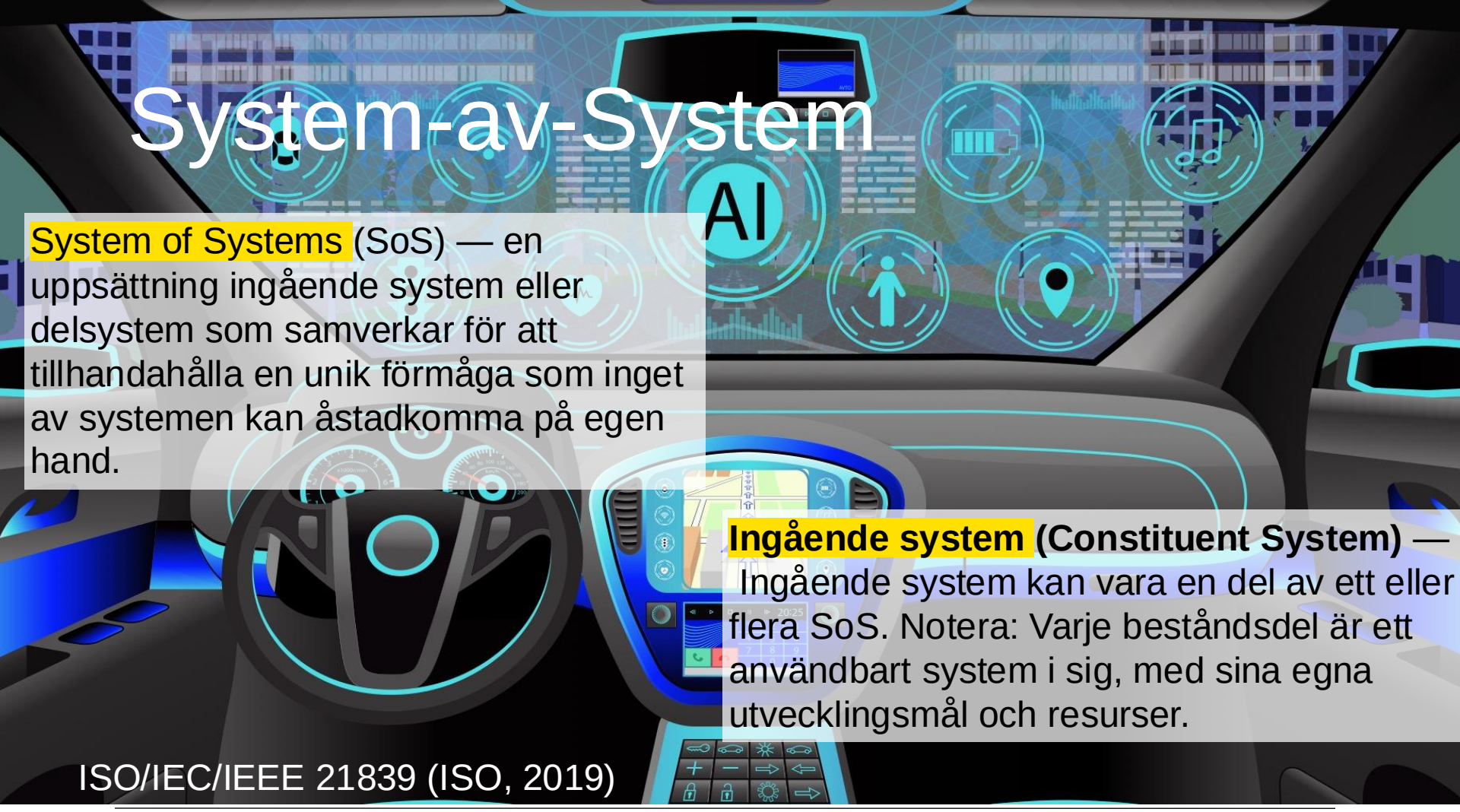
Dagens uppkopplade system ger mer komplexa sammanhang

Sammanhanget utgör omgivningen till mjukvarusystemet.

Sammanhanget ställer krav och avgränsar mjukvarusystemet



System-av-System



System of Systems (SoS) — en uppsättning ingående system eller delsystem som samverkar för att tillhandahålla en unik förmåga som inget av systemen kan åstadkomma på egen hand.

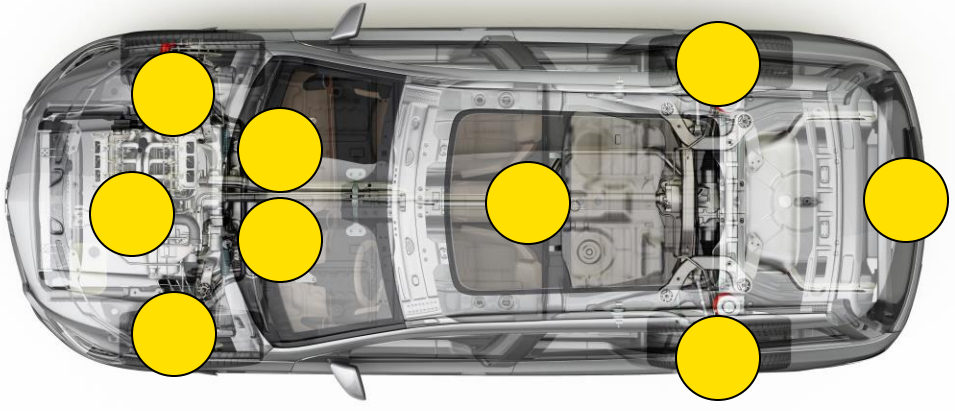
Ingående system (Constituent System) — Ingående system kan vara en del av ett eller flera SoS. Notera: Varje beståndsdel är ett användbart system i sig, med sina egna utvecklingsmål och resurser.

ISO/IEC/IEEE 21839 (ISO, 2019)

System-av-System

System av System (SoS) — samverkar för att tillhandahålla en unik förmåga som inget av de ingående systemen kan åstadkomma på egen hand.

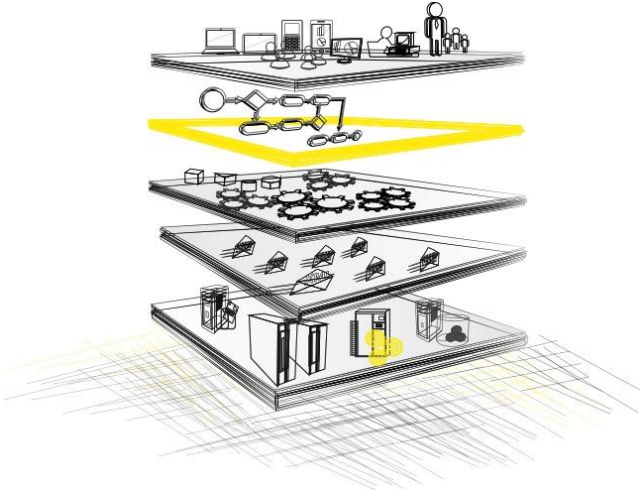
Constituent Systems



ISO/IEC/IEEE 21839 (ISO, 2019)

Organisationsarkitektur

Organisationsarkitekturen omfattar strukturen och beteendet hos en organisation, inklusive dess processer, informationsflöden, personal och organisatoriska enheter. Den kan också innehålla beskrivningar av stödjande IT-infrastruktur, informationsarkitektur och mjukvarusystem.



Organisation

Affärsprocesser och tjänster

Mjukvara

Information

Infrastruktur

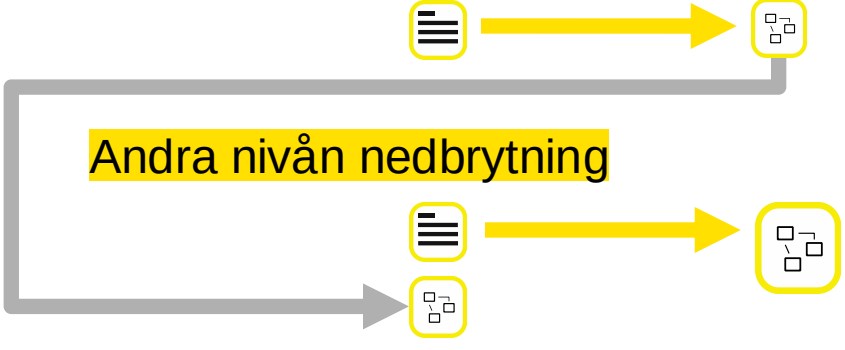
Systemarkitektur

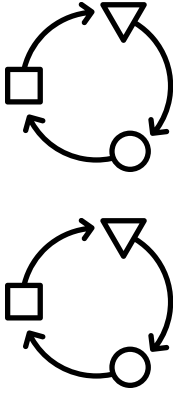
En systemarkitektur är en representation av ett system där det finns en mappning av mjukvaruarkitekturen till hårdvaruarkitekturen, samt hur användare (och andra system) samverkar med dessa

Bass, L., Clements, P., Kazman, R. Software Architecture in Practice 4h ed.

2:a nivån, 3:e och … n:te?

Från svarta till genomskinliga lådor





Lägg till o förfina komponenter

Lägg till o förfina ansvar

Lägg till o förfina interfaces

Lägg till o förfina kopplingar

Dagens takeaways!

Nedbrytning

Abstraktion

Ansvar

Organisationsarkitektur

Mjukvaruarkitektur

Systemarkitektur

2DV604 Software Architecture