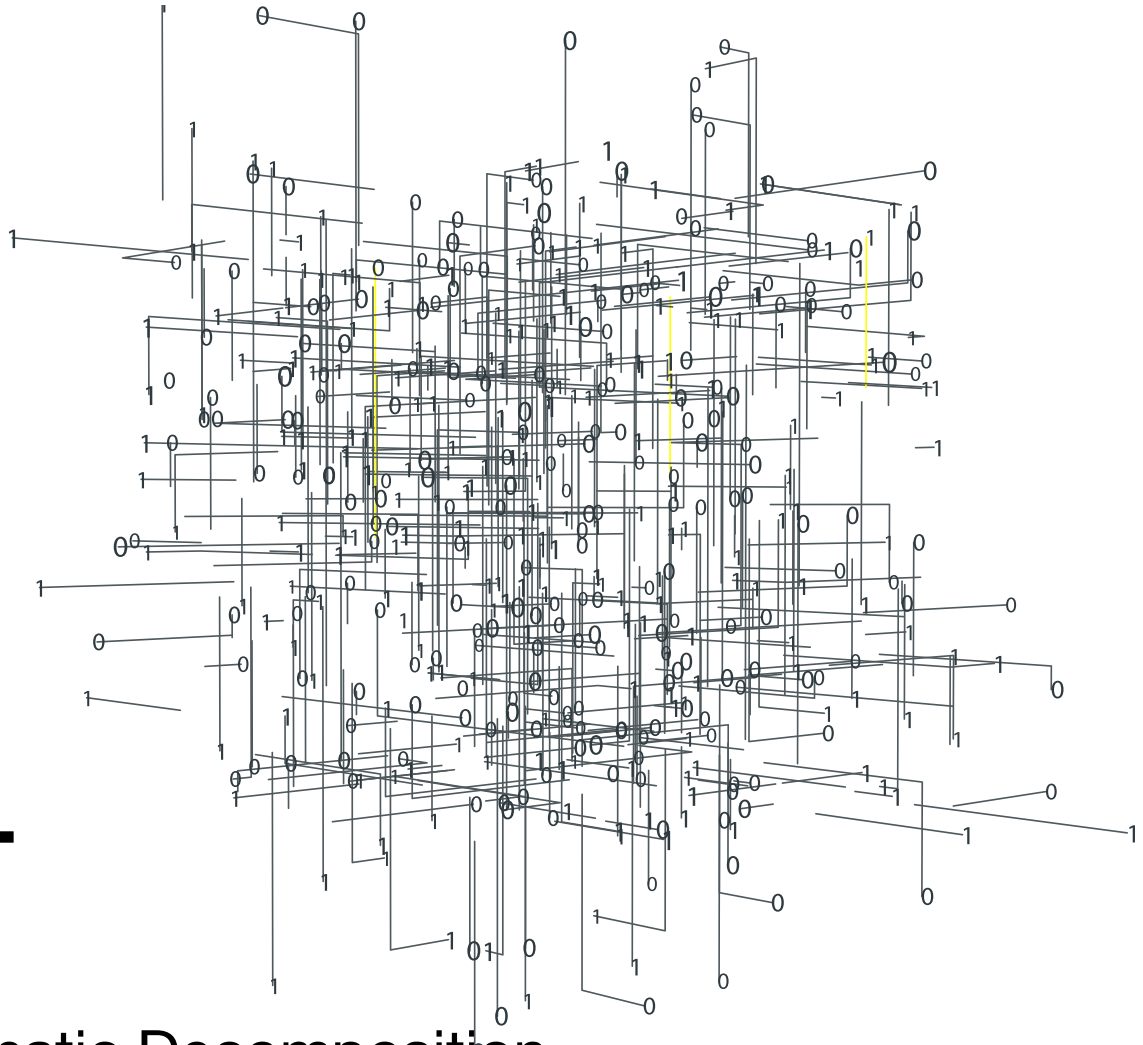


2DV604

Theme 1

Models and Systematic Decomposition

Linnæus University

Todays Lecture

Software Design

Design Practices

Design Quality

Problem Solving

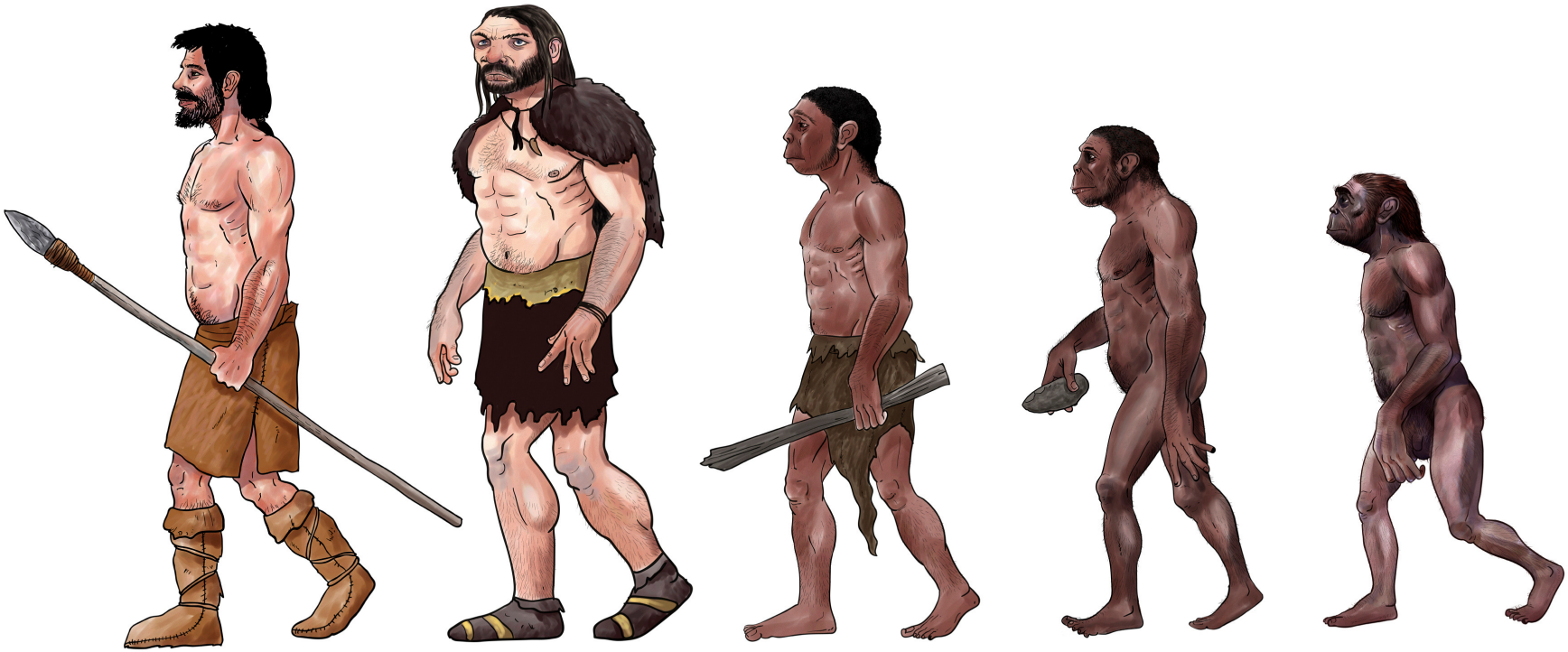
Design Knowledge

Design Principles

‘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).

But what are we doing?



Humans design, create, and use tools!

So, what are we doing?

Polya's problem solving process

1. Understand the problem
2. Devise a plan for solving the problem
3. Carry out the plan
4. Evaluate the result

Software Design is Problem solving with software

The solution is software!

We use software specific methods

Solving Wicked Problems

Solutions cannot be immediately tested

No single explanation for the problem

Solutions are better or worse

No clear definition

No Stopping rules

No alternative solutions

Each problem is connected to another

The solver has no right to be wrong

Solving problem is a one-shot operation

provide interoperable APIs and data models for infrastructure operators, service and device providers

provide a platform for end-users supporting seamless device integration, rule-based automation and configurable UI

No stop rules?

provide a platform for end-users supporting seamless device integration, rule-based automation and configurable UI

“wicked problems”

There is a lack of criteria that can be used to determine when the solution to a problem has been found, so that further work will not be able to improve it.

Solving Wicked Problems

provide interoperable APIs and data models for infrastructure operators, service and device providers

provide a platform for end-users supporting seamless device integration, rule-based automation and configurable UI

Solutions cannot be immediately tested

No single explanation for the problem

Solutions are better or worse

No clear definition

No Stopping rules

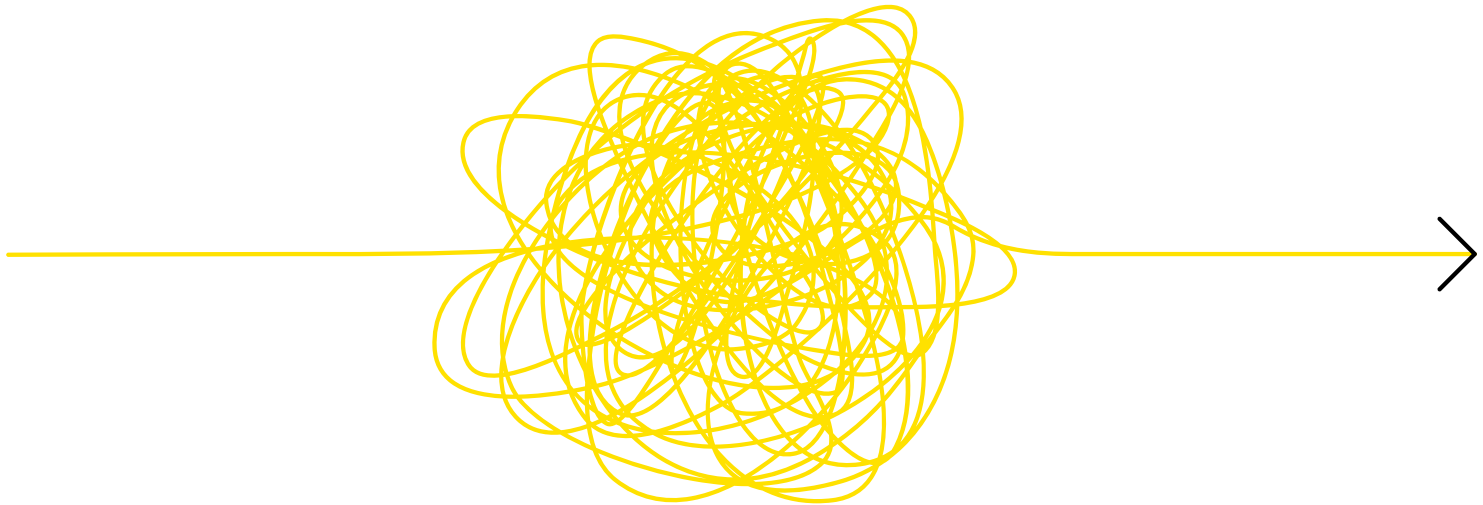
No alternative solutions

Each problem is connected to other

The Solver has no right to be wrong

Solving problem is a one shot operation

Contributes to Problem Complexity



How do we deal with complexity?

How do we Manage Complexity?

Abstraction

- Simplification
- Description of entities with required properties (context dependent).
- Some examples, Data abstraction, Instructions abstraction

Decomposition

- The problem is decomposed in sub problems, which are further decomposed into sub problems …
- The problem divides the solution into parts.
- Divide n' conquer

How do we Manage Complexity?

From Blackbox to White box

- Reduce amount of information
- Don't "open up" to early – a can of worms?
- Iterative and stepwise-refinement

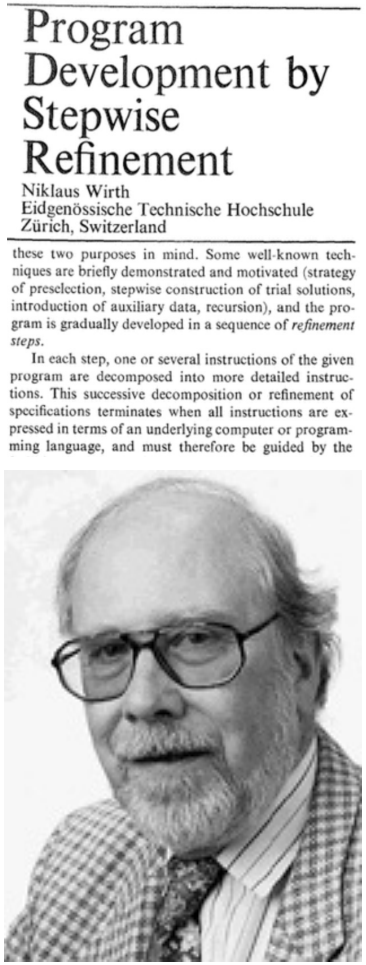
Incremental

- Work on parts of the system
- Reduces problem size

Step-wise Refinement

Recognized as the first generalized software ”design method”

- Program construction consists of a sequence of refinement steps.
- In each step a given task is broken up into a number of suhtasks.
- Each refinement in the description of a task may be accompanied by a refinement of the description of the data which constitute the means of communication between the subtasks.
- Refinement of the description of program and data structures should proceed in parallel.



Program Development by Stepwise Refinement

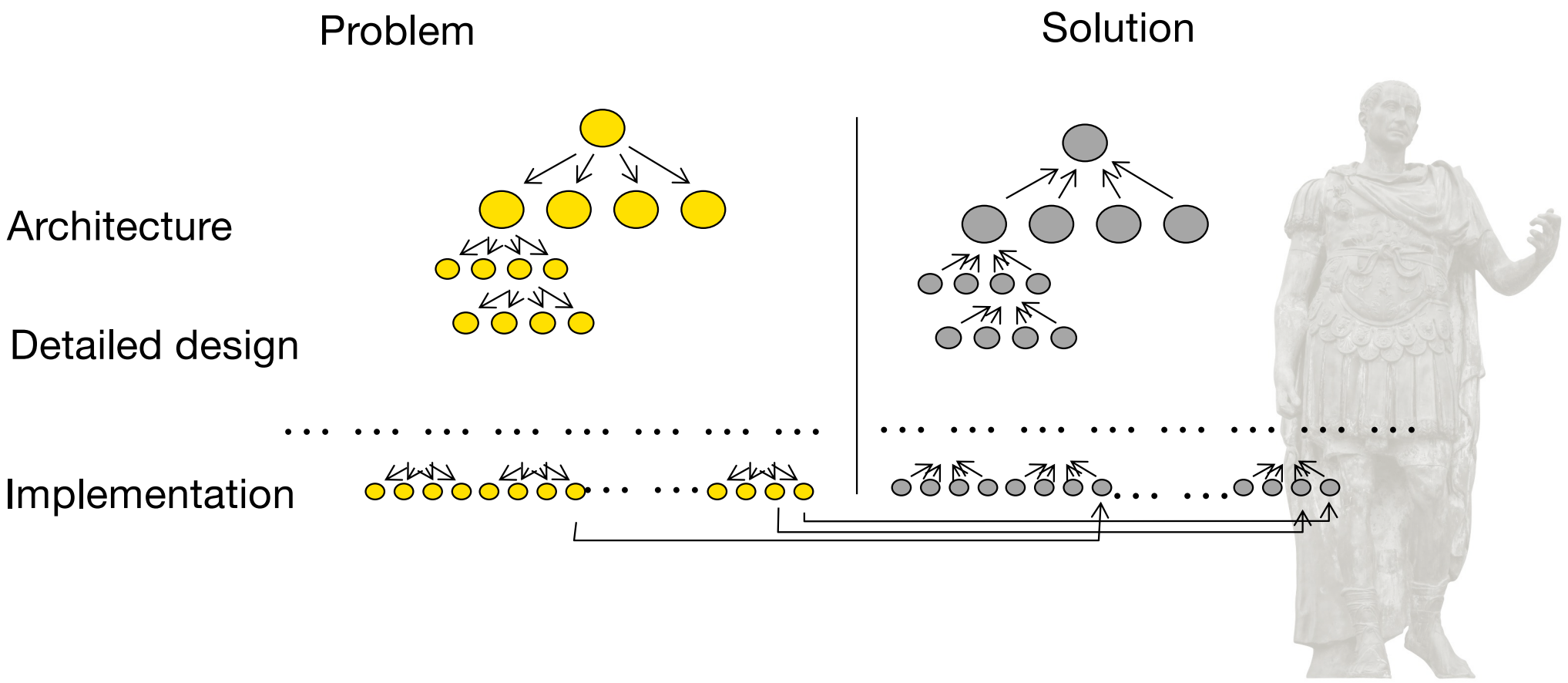
Niklaus Wirth
Eidgenössische Technische Hochschule
Zürich, Switzerland

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

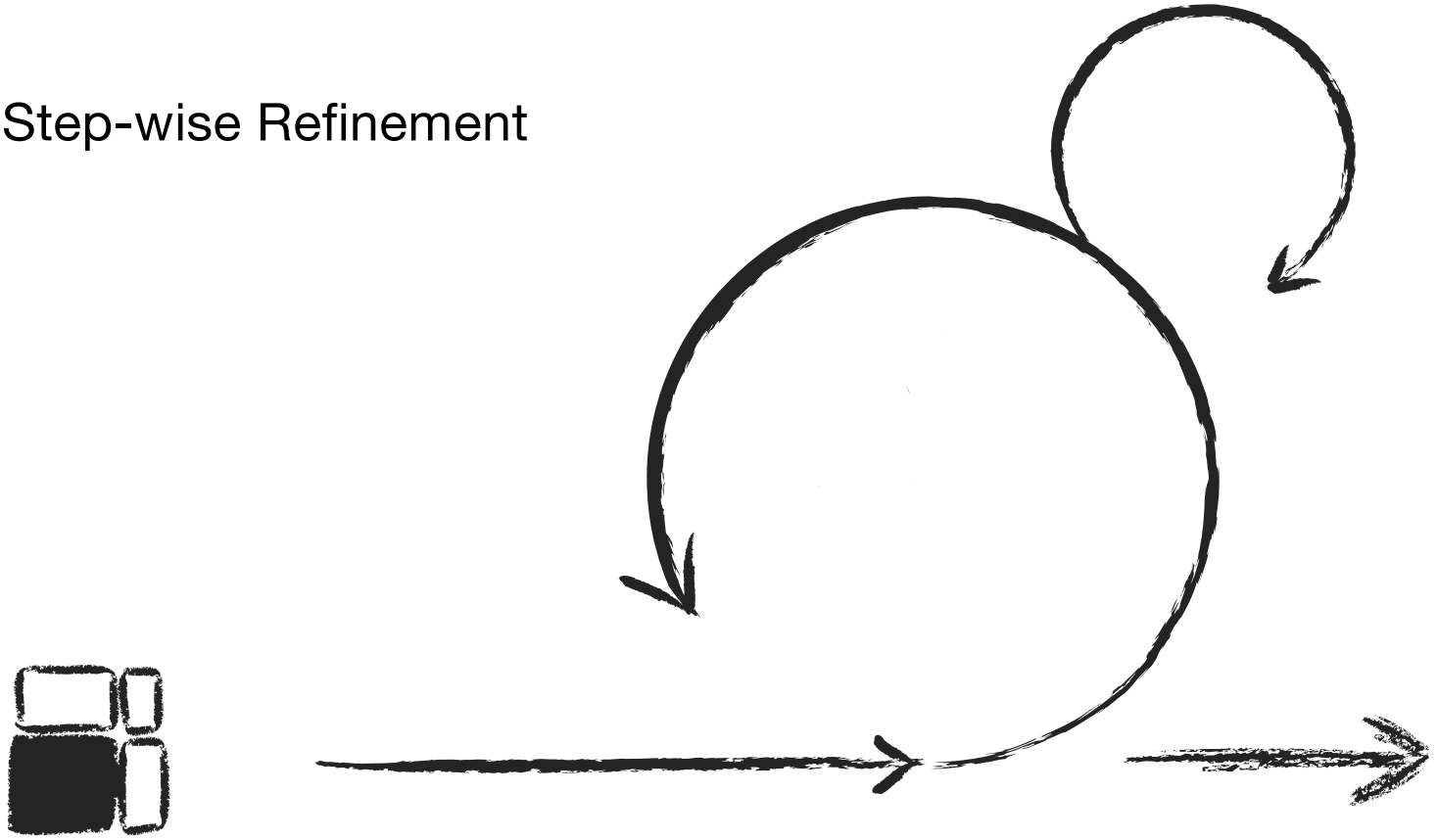
Niklaus Wirth

Design Practice – Divide n’ Conquer



Iterative Development

Step-wise Refinement



Small tasks – Small checks – Small changes

Incremental Development

Architectural models

Motivation

The Role of Models

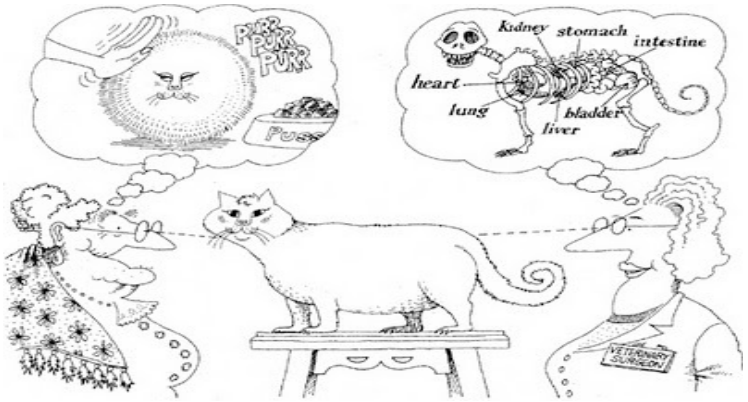
Models support communication

We establish a shared language with a vocabulary and semantic

Models are constructed to enable reasoning within an idealized logical framework about these processes.

Idealized means that the model may make explicit assumptions that are known to be false in some detail → simplifications!

Abstraction



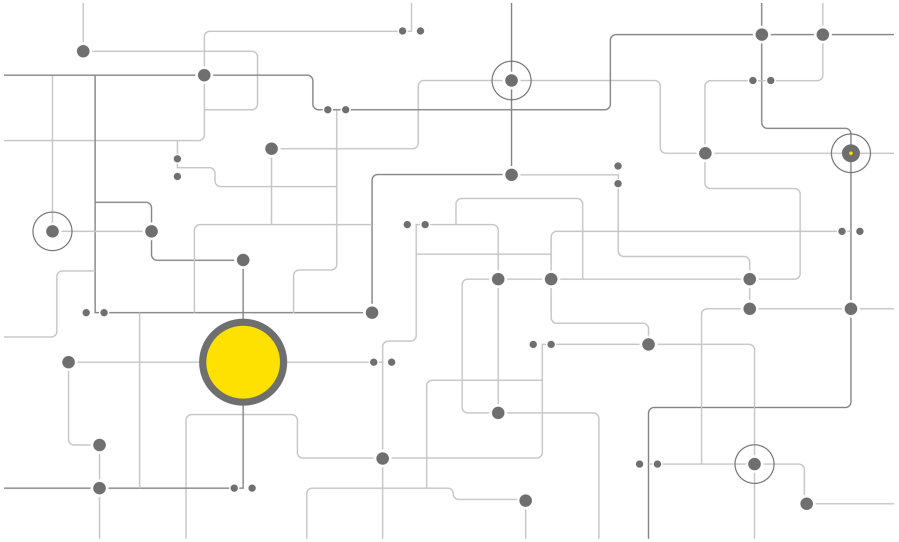
"a ***simplified*** description, or specification, of a system that ***emphasizes some*** of the system's ***details*** or properties ***while suppressing others***.

A good abstraction is one that emphasizes details that are significant to the reader or user and suppresses details that are, at least for the moment, immaterial or diversionary." – *Shaw, Mary. 1984*

Modularity

Divide a system into parts

Modularity in software development is a measure of how divided a system is into parts (modules or components). Each part has a specific responsibility (functionality) and works independently to the greatest



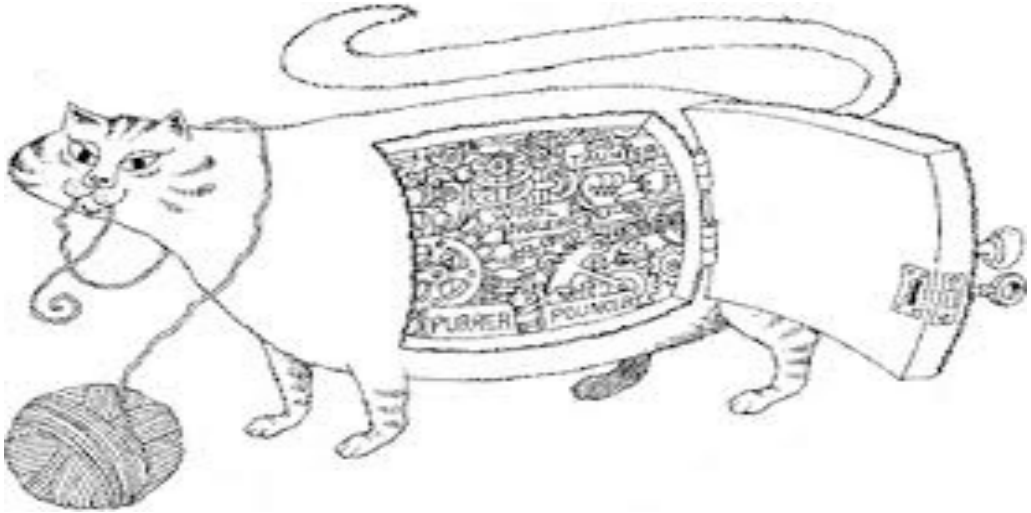
Hierarchy

Establish hierarchies – Compose subsystems into larger systems

Whole – Parts

Generalization – Specialization

Encapsulation – Information Hiding



"the process of compartmentalizing the elements of an abstraction that constitute its structure and behavior; encapsulation serves to separate the contractual interface of an abstraction and its implementation."

"control direct access to structure and behavior so that only what is in the open contract is accessible from other abstractions"

Design Principles

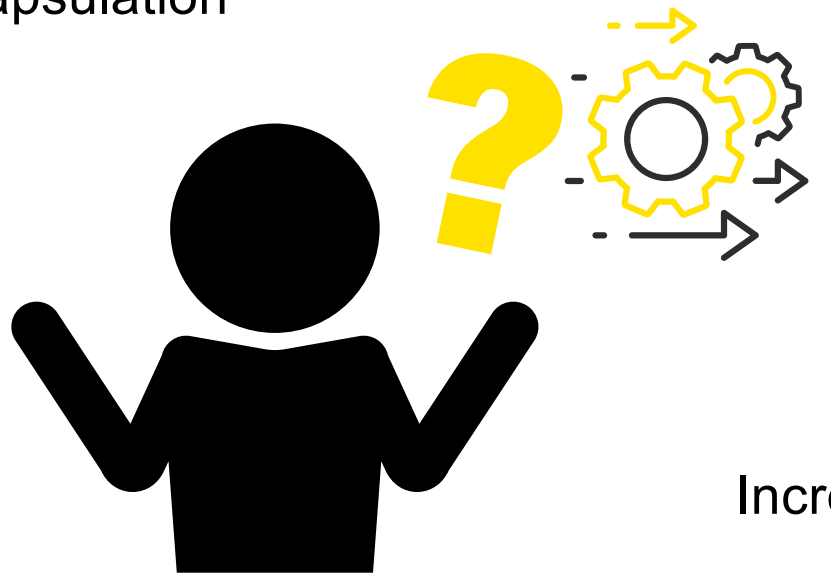
from Black Box to White Box solving wicked problems

Putting it all together

Information Hiding

Encapsulation

Abstraction



Hierarchy

Modularity

Iterative

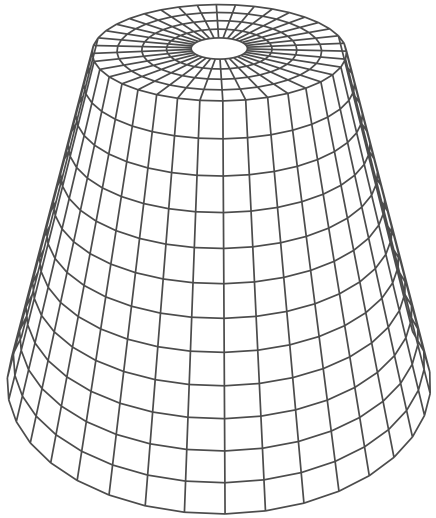
Incremental

Partitioning – Decomposition

Modularity and Hierarchy

From Black-box to White box

1. Partition the problem
2. Parts and Responsibilities
3. Divide n’ conquer



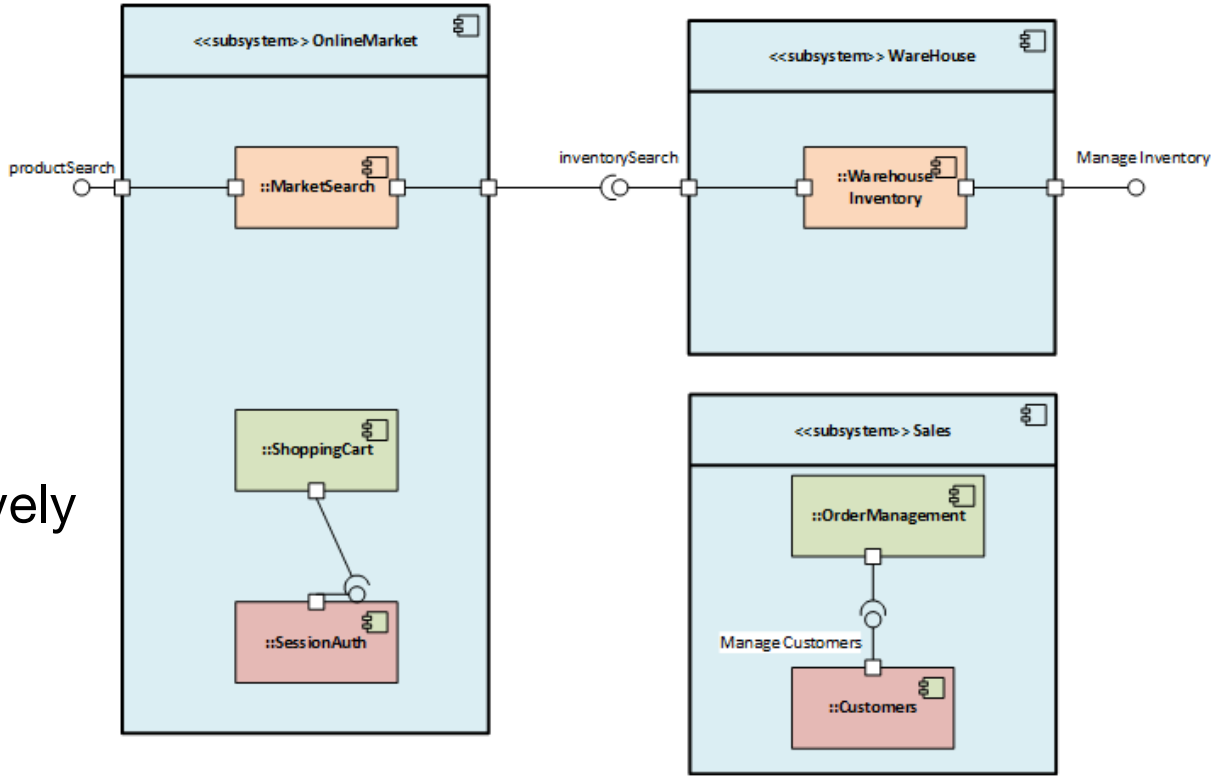
Perspectives

1. Layering or distribution
2. Functionality – interfaces
3. Separation of concerns

Responsibilities

Allocate Responsibilities to Abstractions

- Functionality – Interfaces
- Services



Black box to White-box

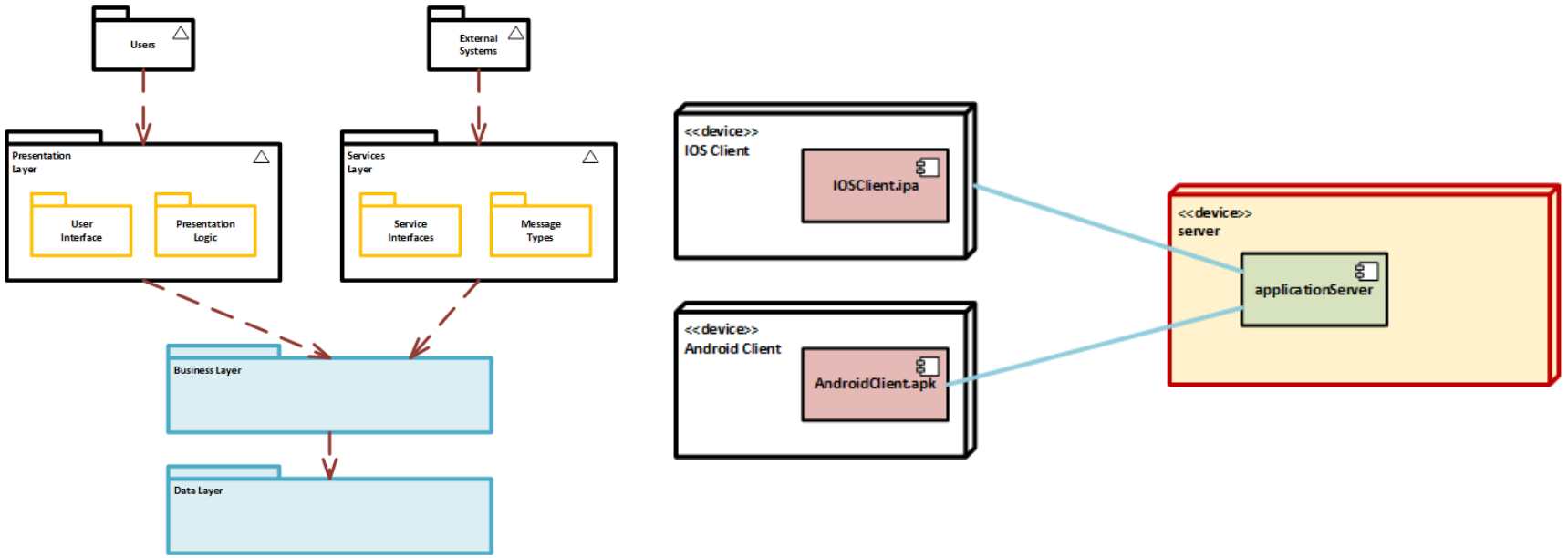
– Start at the highest of levels

– Refine responsibilities iteratively

Perspective

Layers and Distribution

- Layers – hide complexity (logical)
- Distribution – computational partitioning (physical)
- Allocate Responsibilities



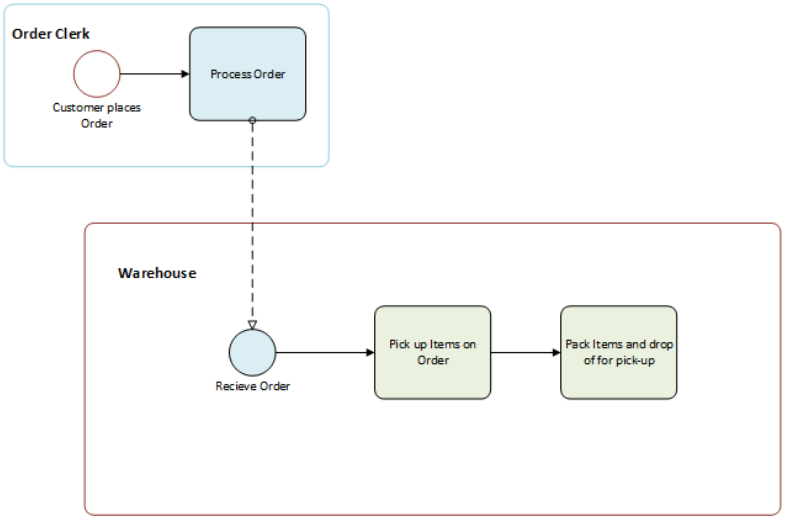
Perspective

Business Processes

Responsibilities

Processes

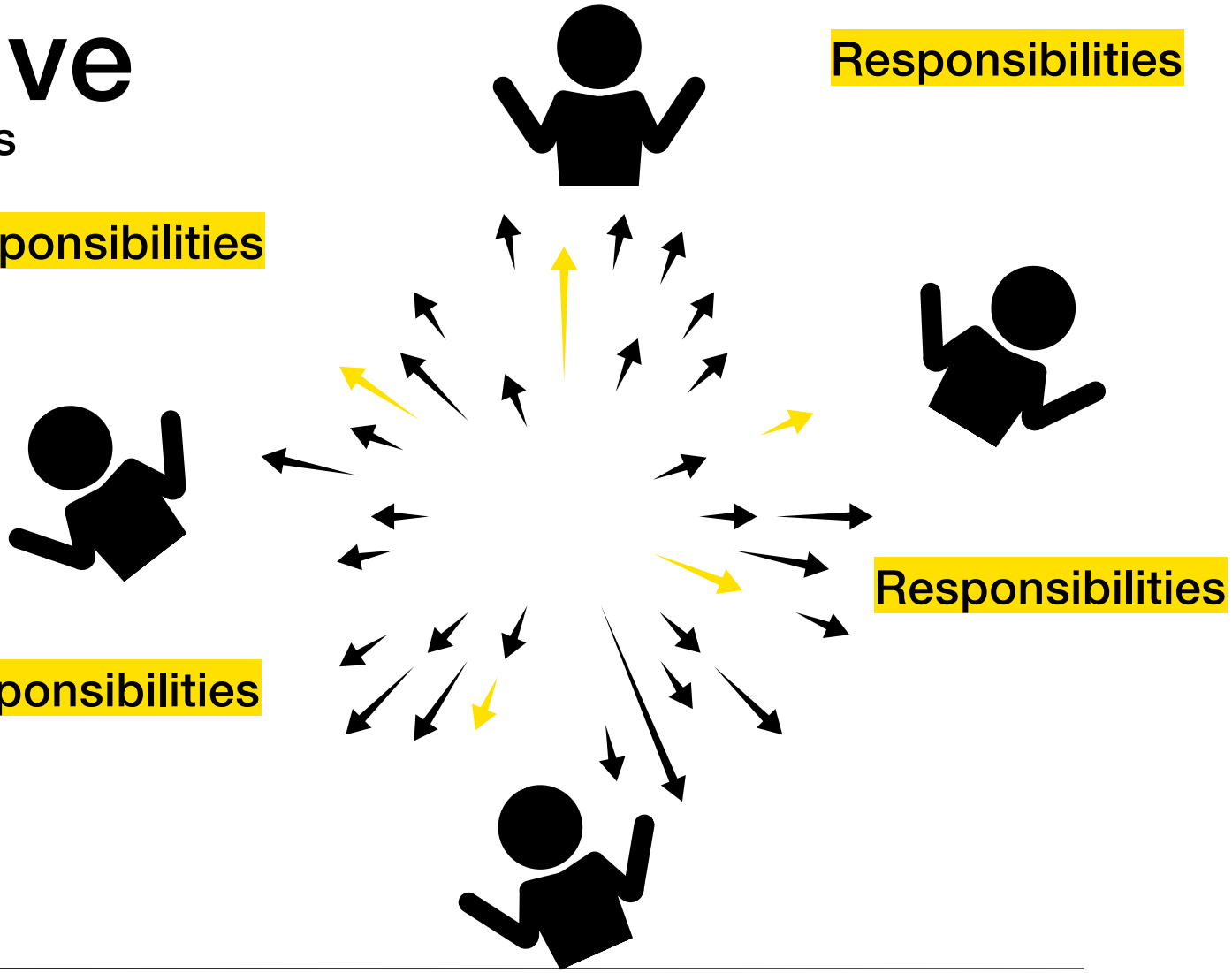
I. Customer places order over the telephone with credit card
II. Order clerk charges credit card
III. Clerk notes down SKU and customer details
IV. Clerk emails SKU and details to the warehouse
V. Item is packaged up with label
VI. Shipping mail it to the customer

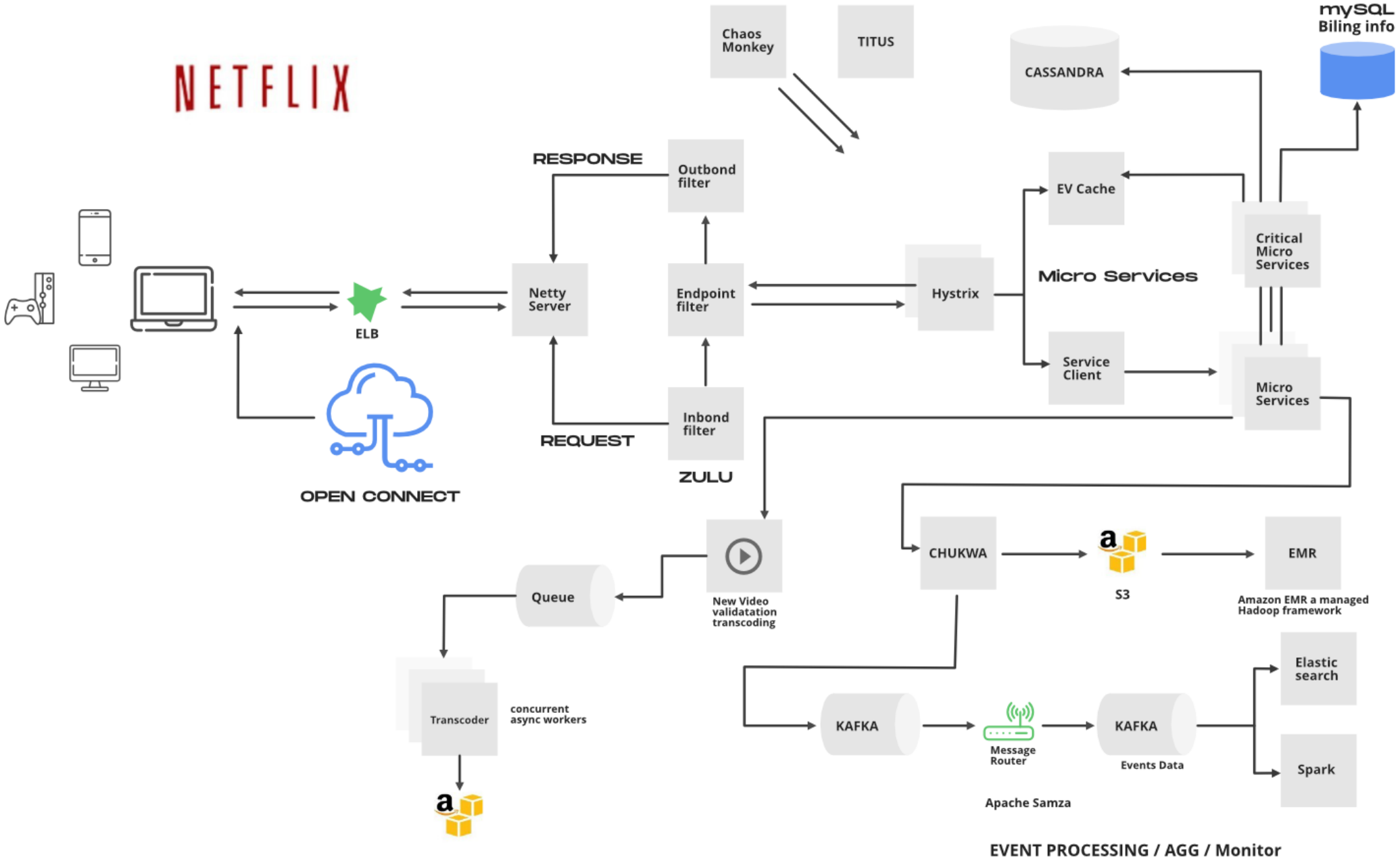


Perspective

Separation of Concerns

functionality
security
performance
safety
persistency
availability
etc.




NETFLIX
Chaos Monkey
TITUS
CASSANDRA
mySQL
Biling info
RESPONSE
Outbond filter
EV Cache
Critical Micro Services
Micro Services
Hystrix
Netty Server
Endpoint filter
ELB
Service Client
Micro Services
Inbond filter
REQUEST
ZULU
OPEN CONNECT
CHUKWA
S3
EMR
Amazon EMR a managed Hadoop framework
Queue
New Video validatation transcoding
Transcoder
concurrent async workers
KAFKA
Message Router
KAFKA
Events Data
Elastic search
Spark
Apache Samza
EVENT PROCESSING / AGG / Monitor

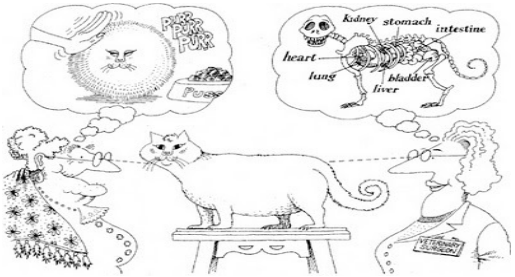
Design purpose

Be clear about the purpose of the design.

Why are you doing this architectural design?

- Decompose (divide) the functional responsibilities
- An abstract system overview
- Test a possible solution
- Describe part of a system (or the whole) when development is underway.

1st level decomposition



untangles the complexity

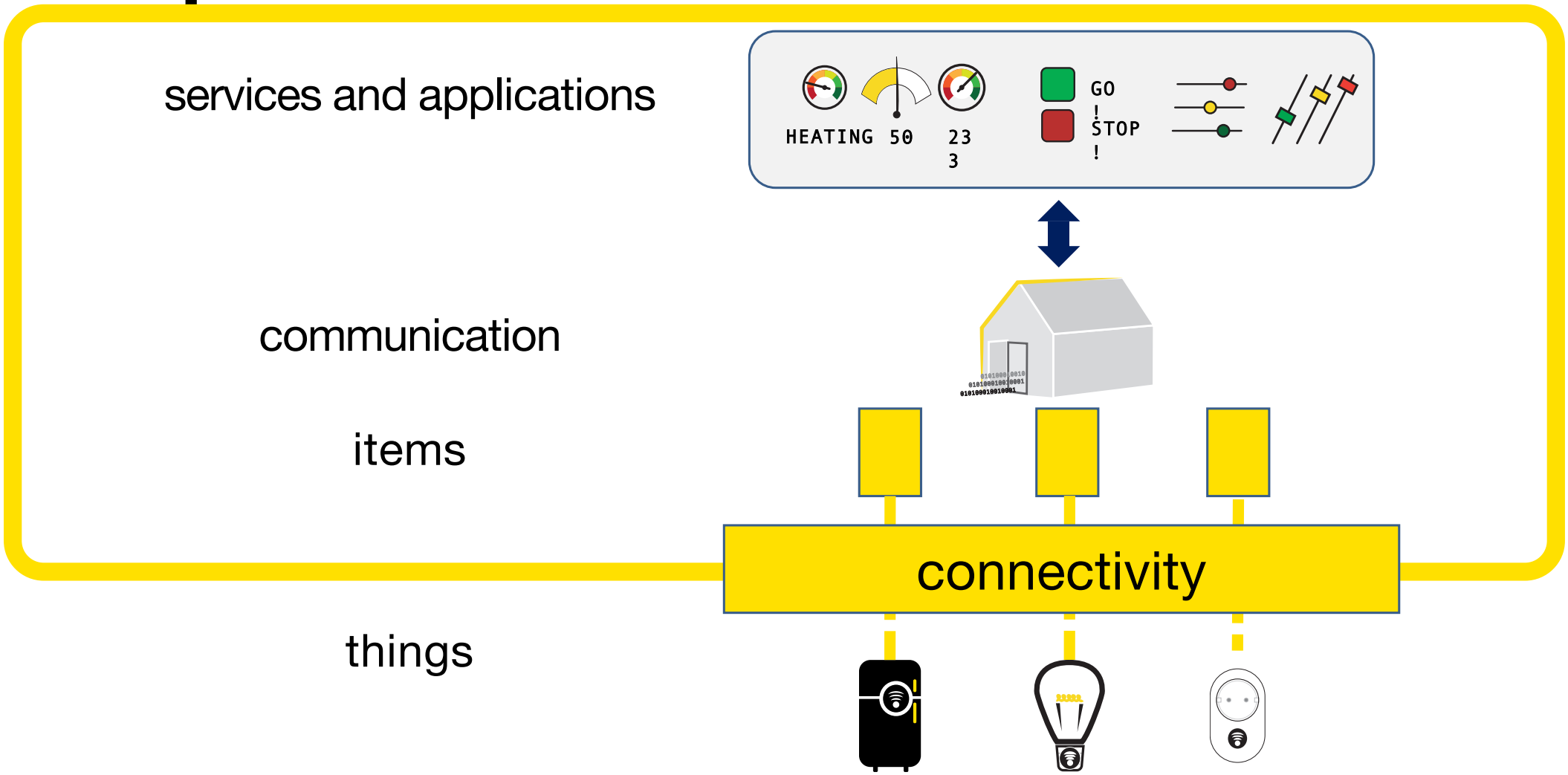
tools for simplification

- **abstraction**
- modularity
- hierarchy
- encapsulation

"a ***simplified*** description, or specification, of a system that ***emphasizes some*** of the system's ***details*** or properties ***while suppressing others***.

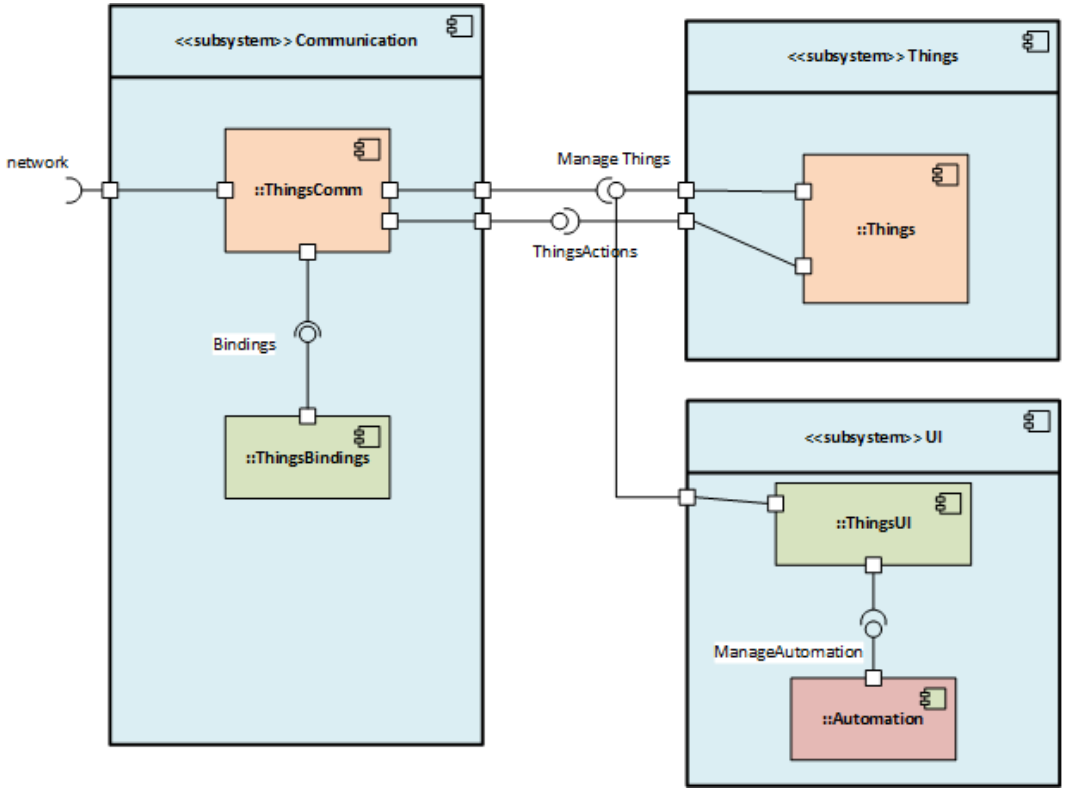
A good abstraction is one that emphasizes details that are significant to the reader or user and suppresses details that are, at least for the moment, immaterial or diversionary." -- Shaw, M. 1984

openHAB 1st level abstract



openHAB

Functional

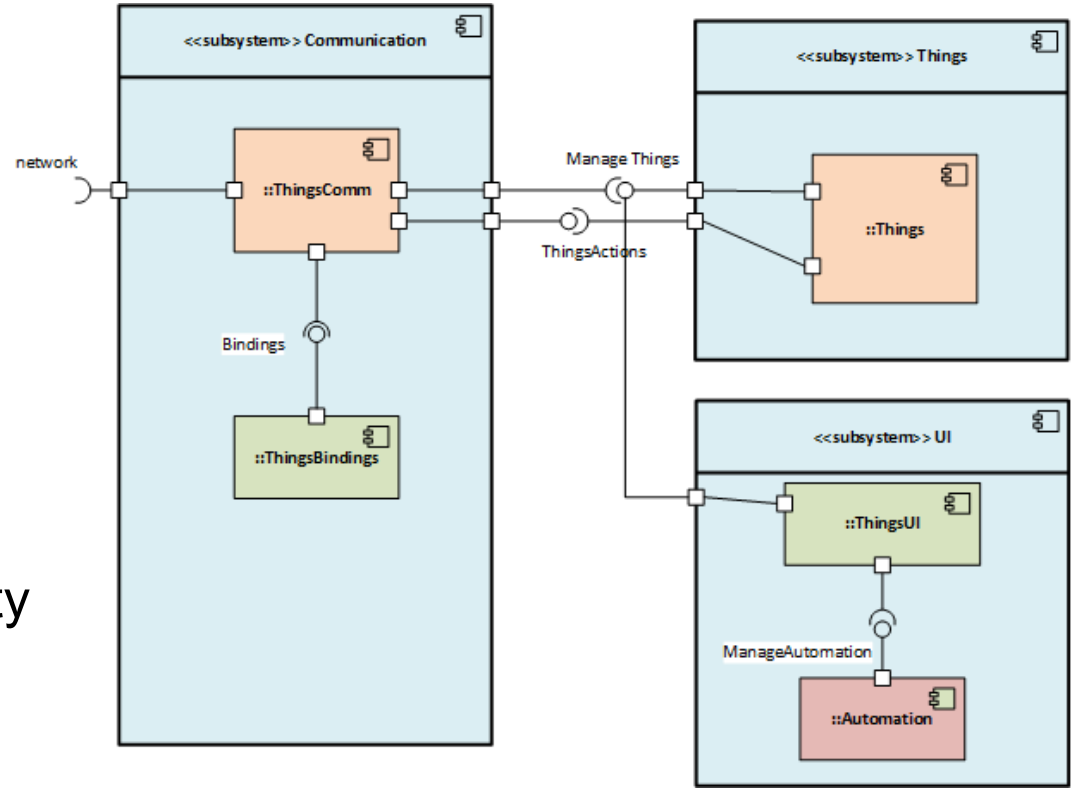


openHAB

Functional

hierarchy

modularity



abstraction

encapsulation

Works in the small and LARGE

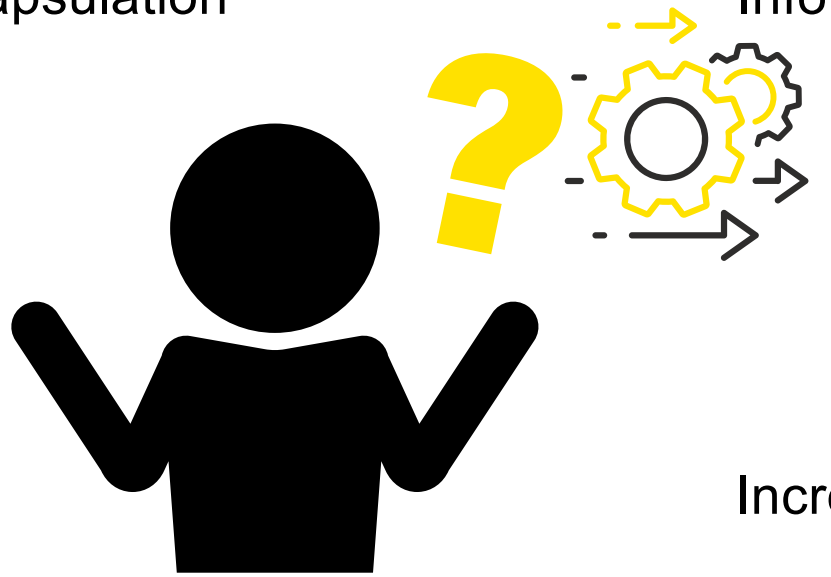
From Black Box to White Box solving wicked problems

Decompose

Encapsulation

Information Hiding

Abstraction



Hierarchy

Modularity

Iterative

Incremental

Perspectives

Software Architecture

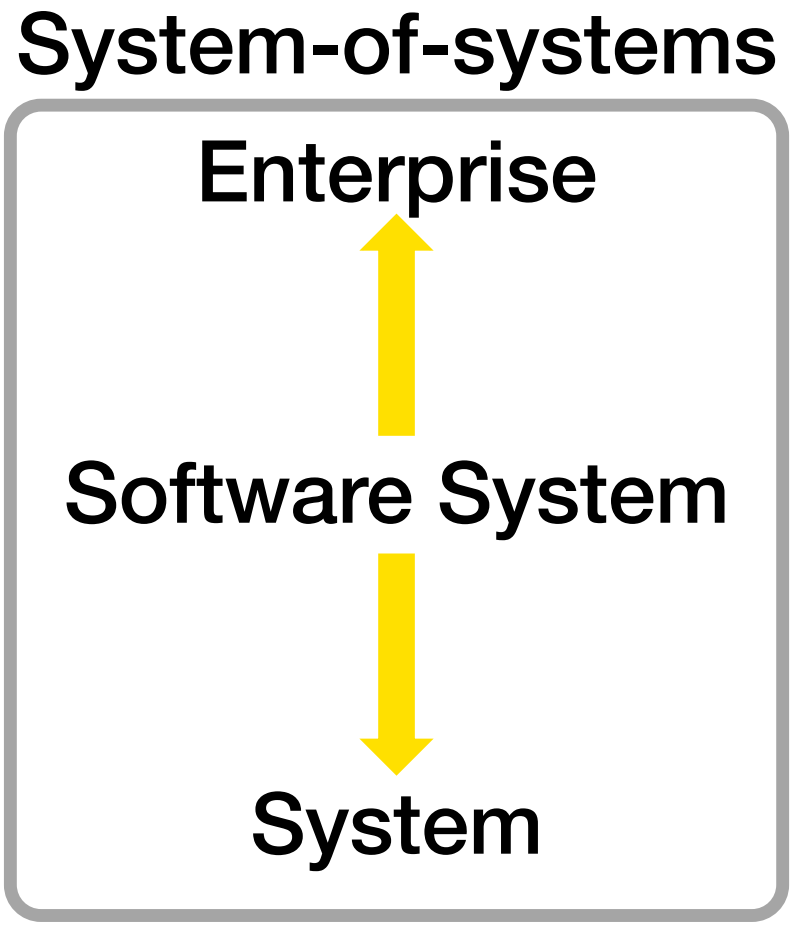
Context

Software exists in a context.

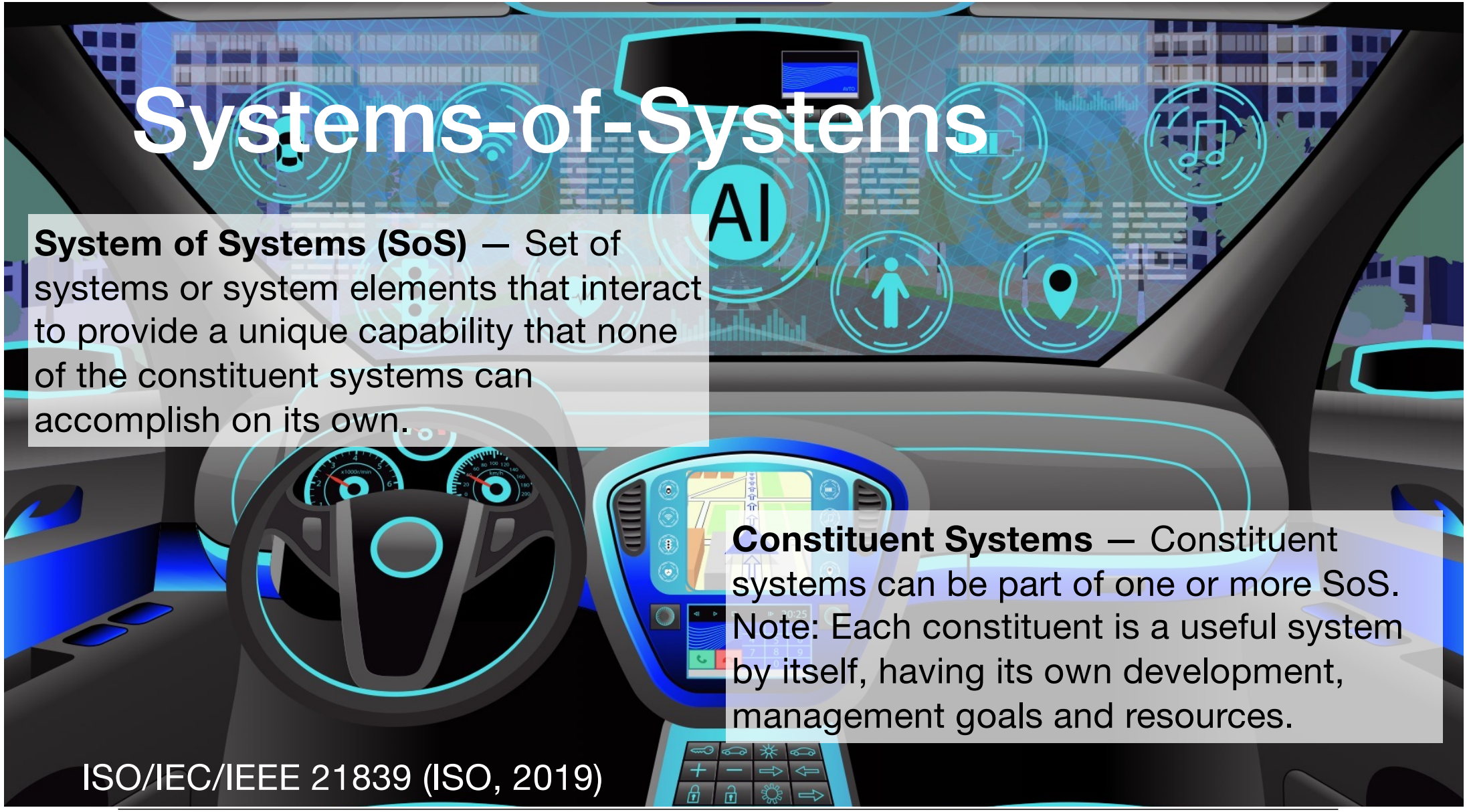
Today's connected systems have much more complex contexts.

The context defines an environment for the software system.

The context sets constraints and requirements for the software.



Systems-of-Systems



System of Systems (SoS) — Set of systems or system elements that interact to provide a unique capability that none of the constituent systems can accomplish on its own.

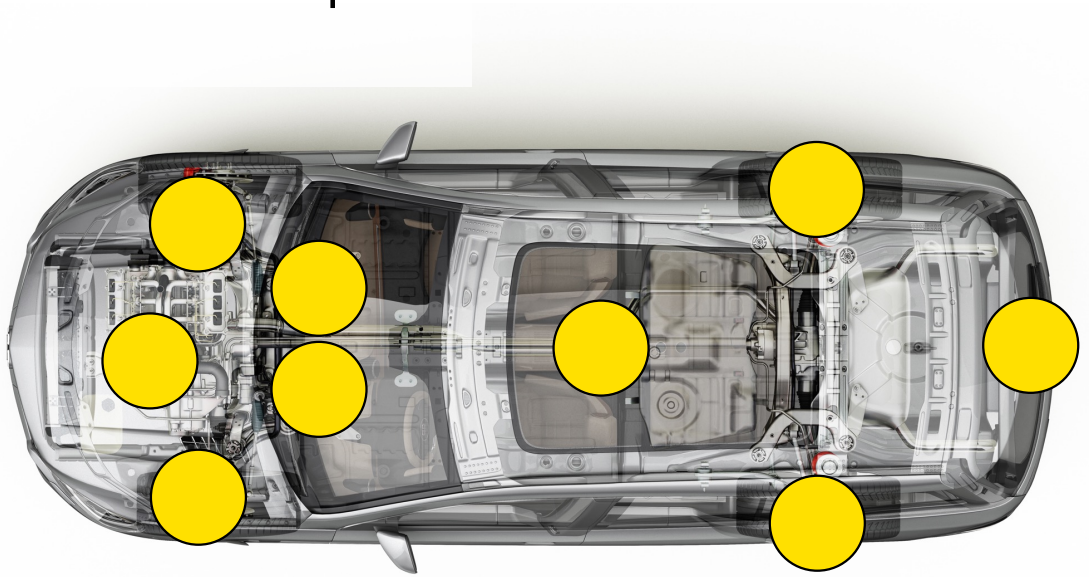
Constituent Systems — Constituent systems can be part of one or more SoS.
Note: Each constituent is a useful system by itself, having its own development, management goals and resources.

ISO/IEC/IEEE 21839 (ISO, 2019)

Systems-of-Systems

System of Systems (SoS) — interact to provide a unique capability that none of the constituent systems can accomplish on its own.

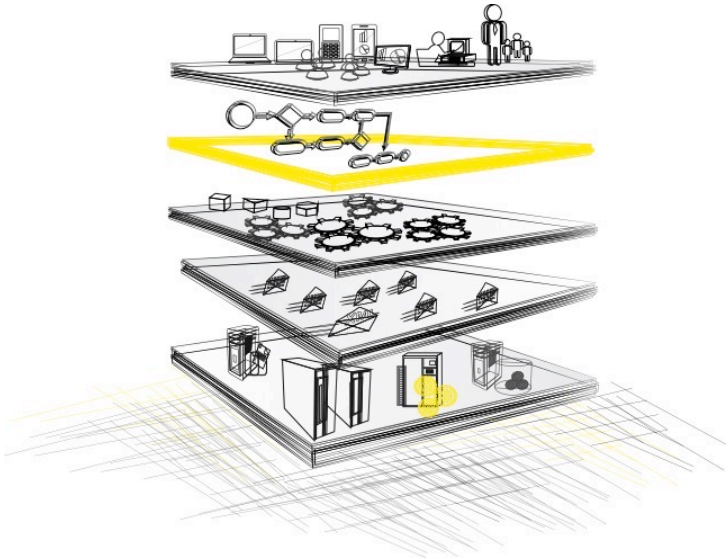
Constituent Systems



ISO/IEC/IEEE 21839 (ISO, 2019)

Enterprise Architecture

Enterprise architecture captures the structure and behavior of an organization, including its processes, information flow, personnel, and organizational subunits. It may also include descriptions of supporting IT infrastructure, information architecture and software systems.



Organization

Business and services

Software

Information

Infrastructure

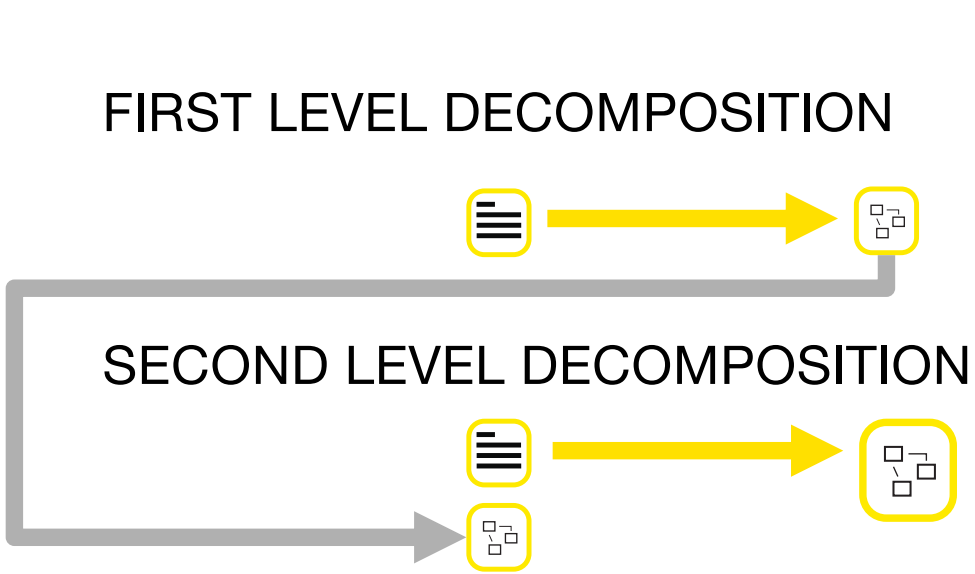
System Architecture

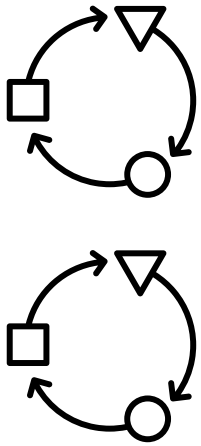
A system's architecture is a representation of a system in which there is a mapping of functionality onto hardware and software components, a mapping of the software architecture onto the hardware architecture, and a concern for the human interaction with these components.

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

2nd level and 3rd and … nth?

From Black-box to White-box





Add and refine components

Add and refine responsibilities

Add and refine interfaces

Add and refine connections

Todays takeaways!

Decomposition

Abstraction

Responsibility

Organizational architecture

Software architecture

System architecture

2DV604 Software Architecture

Lnu.se