3. Without Knowledge: First Examples

Aims

- Three examples: essential mistakes
- Mistakes are wanted: to show concepts of the lecture
- Mistakes are often found in industrial solutions

Later, we come back to these examples. We show, how their architecture should be modeled

Contents of Chapter 3

The Task

We regard the central part of a simple telegram accounting system.

Different telegrams arrive as one text stream. We compute a list containing the number if words and the price of each telegram. Furthermore, the total number of words of all telegrams and the total price is calculated.

1. The input is given by a regular expression
   $(d^6\ b^+\ (c^+b^+)^+\ \texttt{STOP}\ b^+\ \texttt{STOP}\ b^+)^+$
   where d is a decimal digit, b the blank character, c any graphic character. d^6 is the telegram identification code (TID), c^+ are the words contained in the telegram, including the sentence delimiter STOP. The character sequence STOP b^+ STOP is the end of a telegram.
2. The output of the program is a list of triples, sorted according to increasing TIDs,
   (TID, no_words, price_telegram)
   where the price is calculated as follows:
   A word costs 50 ct, a telegram more or equal 3 €. A word is a sequence of characters without the blank character, which is enclosed by sequences of at least one blank. A word is calculated as multiple “words” if it contains more than 12 characters (13 characters = 2 “words”, 25 characters = 3 “words” etc.)
   The separation string STOP is not counted.

Example Input / Output

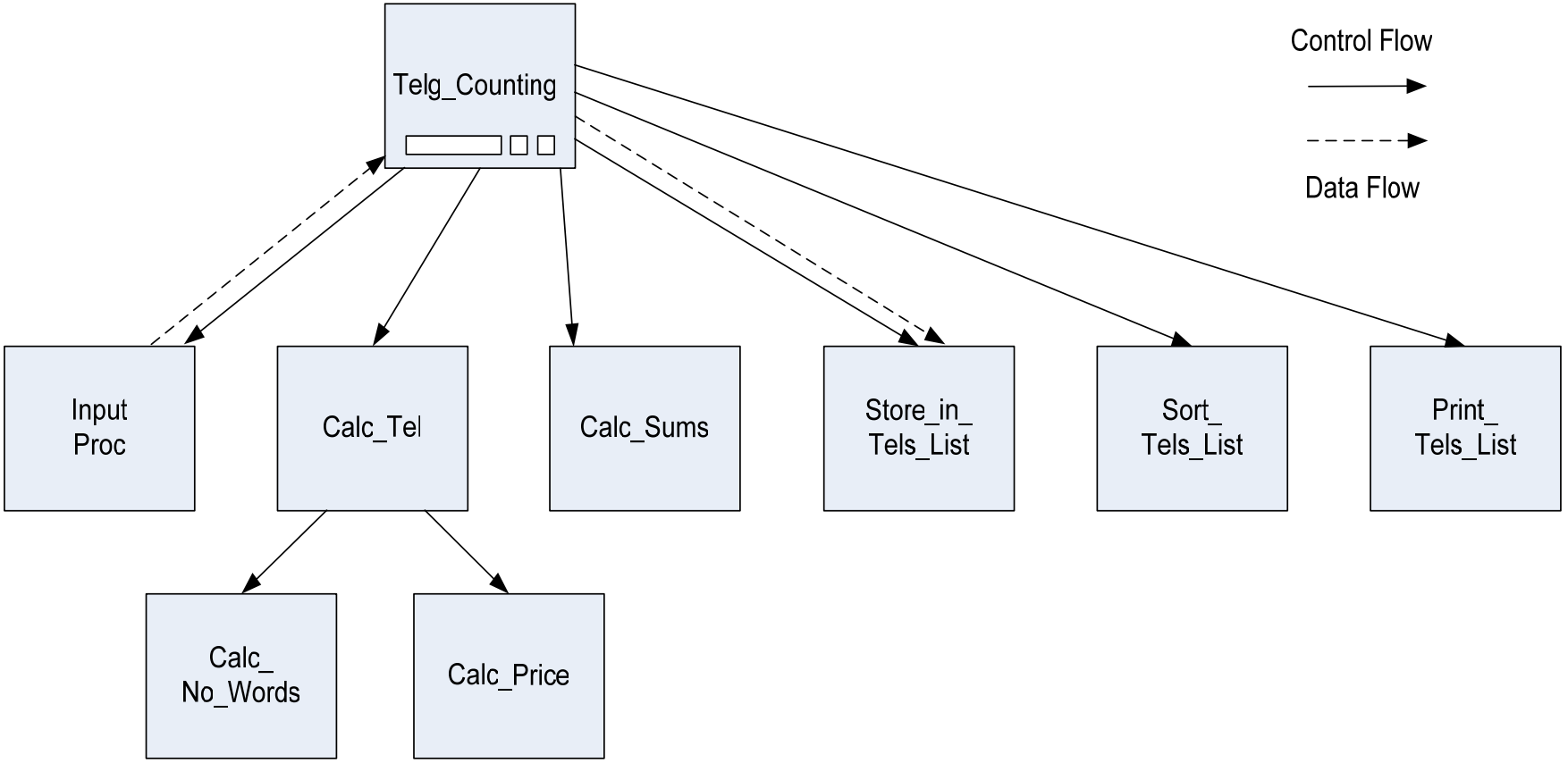
Input

```
635444 HI STOP I COME ON MONDAY STOP JOHN STOP STOP 635560 DEAR
JOHN STOP I AM STILL WAITING FOR YOUR DOCUMENTATION STOP SUSAN
STOP STOP 672134 THERE ARE SEVERE PROBLEMS IN YOUR EXPLANATION
STOP ALFREDLONGNAME STOP STOP 673550 OK JB STOP STOP
```

Output

TID Code	Counted Words	Price
635444	6	3.00
635560	11	5.50
672134	9	4.50
673550	2	3.00
	28	16.00

First "Solution"



Characterization

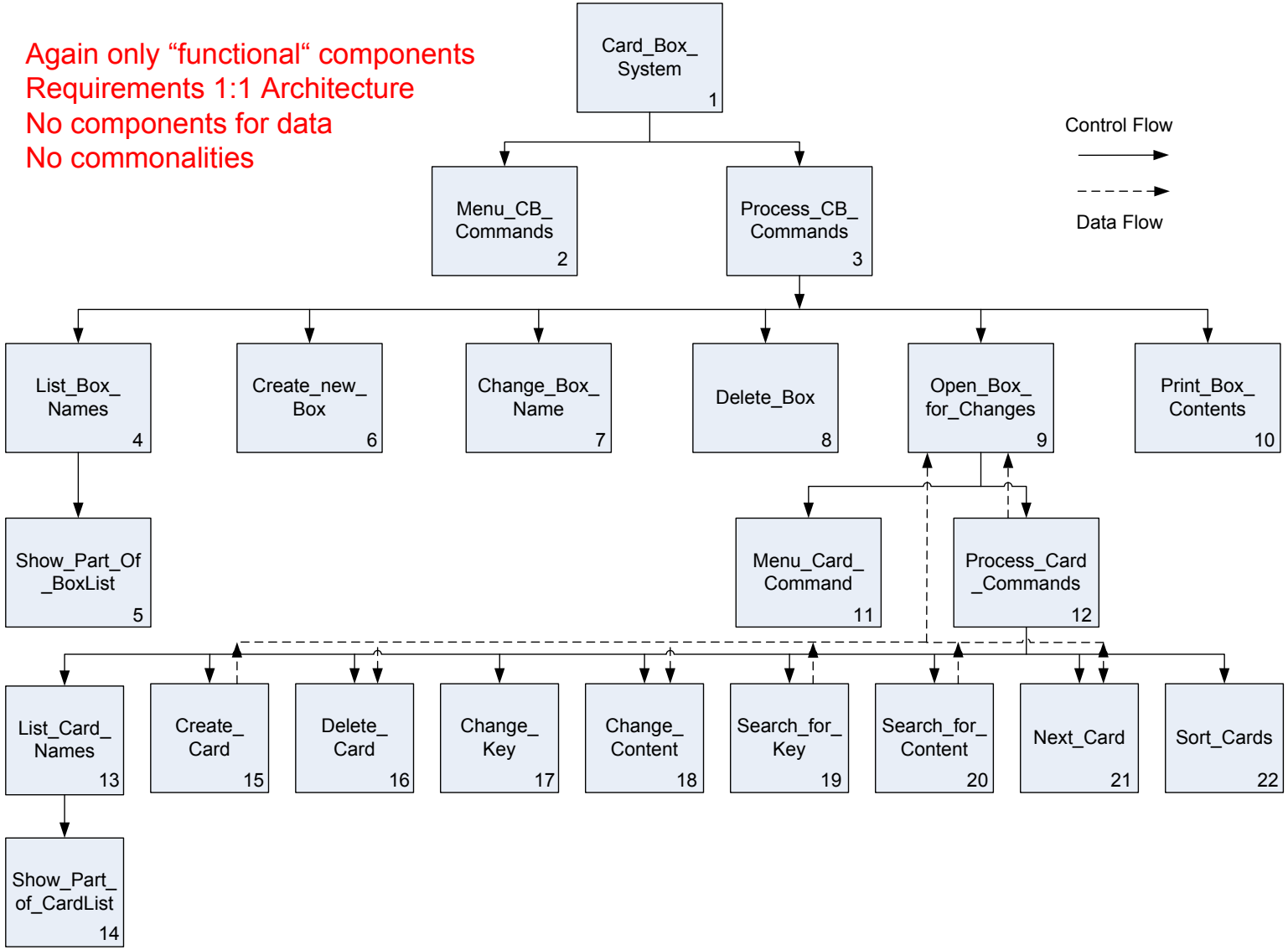
- Structure of “Solution”
  - » Only functional components → subfunction relation
  - » Architecture is tree
    - ⇨ Top-down development
    - ⇨ Missing components
    - ⇨ Thinking only in functional abstraction
  - » Tree is partially ordered
  - » Required tasks match architecture 1:1
  - » Global data, data flows
- Mistakes
  - » Commonalities not recognized, e.g. telegram syntax analysis
  - » Components of very different complexity
    - ⇨ Calc_Sums is a two lines program
    - ⇨ Input_Processing more complex, includes syntax analysis
  - » No components for data

A Card Box System: Requirements

- Boxes of cards
- For bibliography administration
  members of a sports club
  etc.
- Commands on box level
  card level

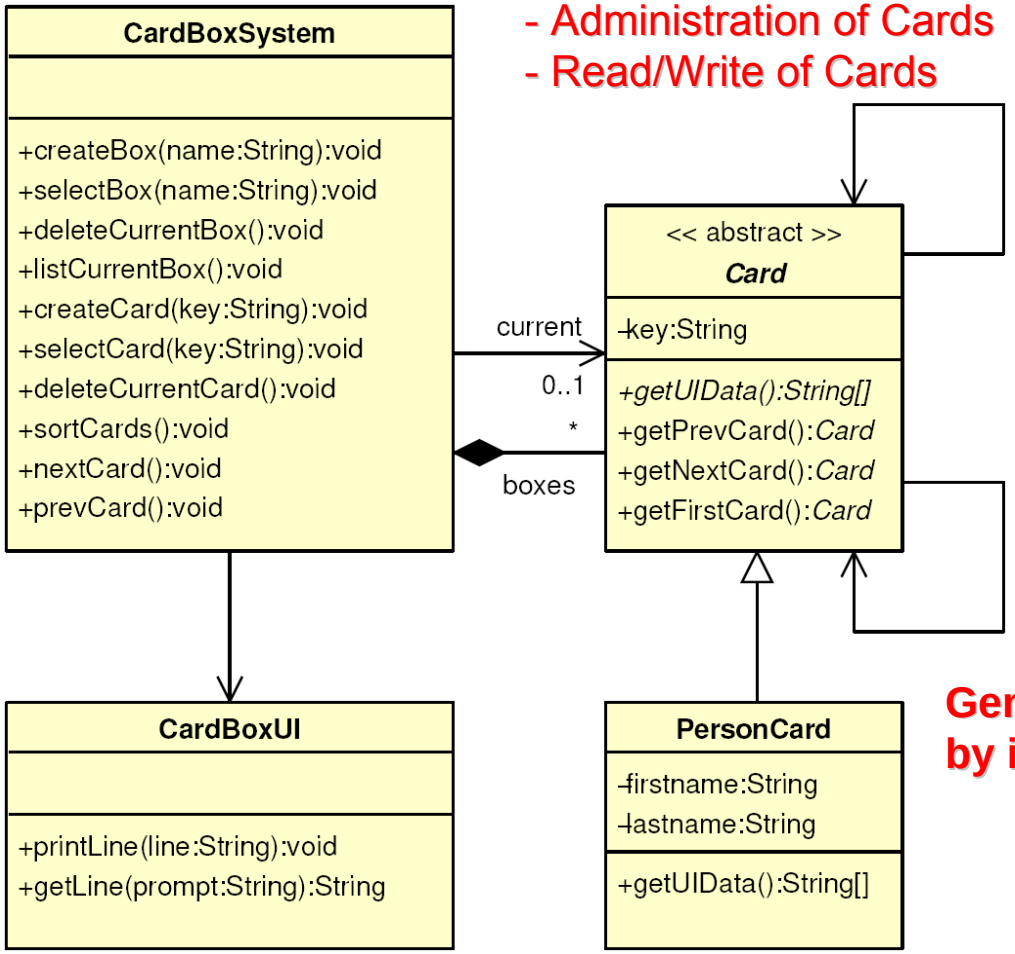
A Card Box System: First Solution

Again only “functional“ components
Requirements 1:1 Architecture
No components for data
No commonalities



Card Box System: Now OO

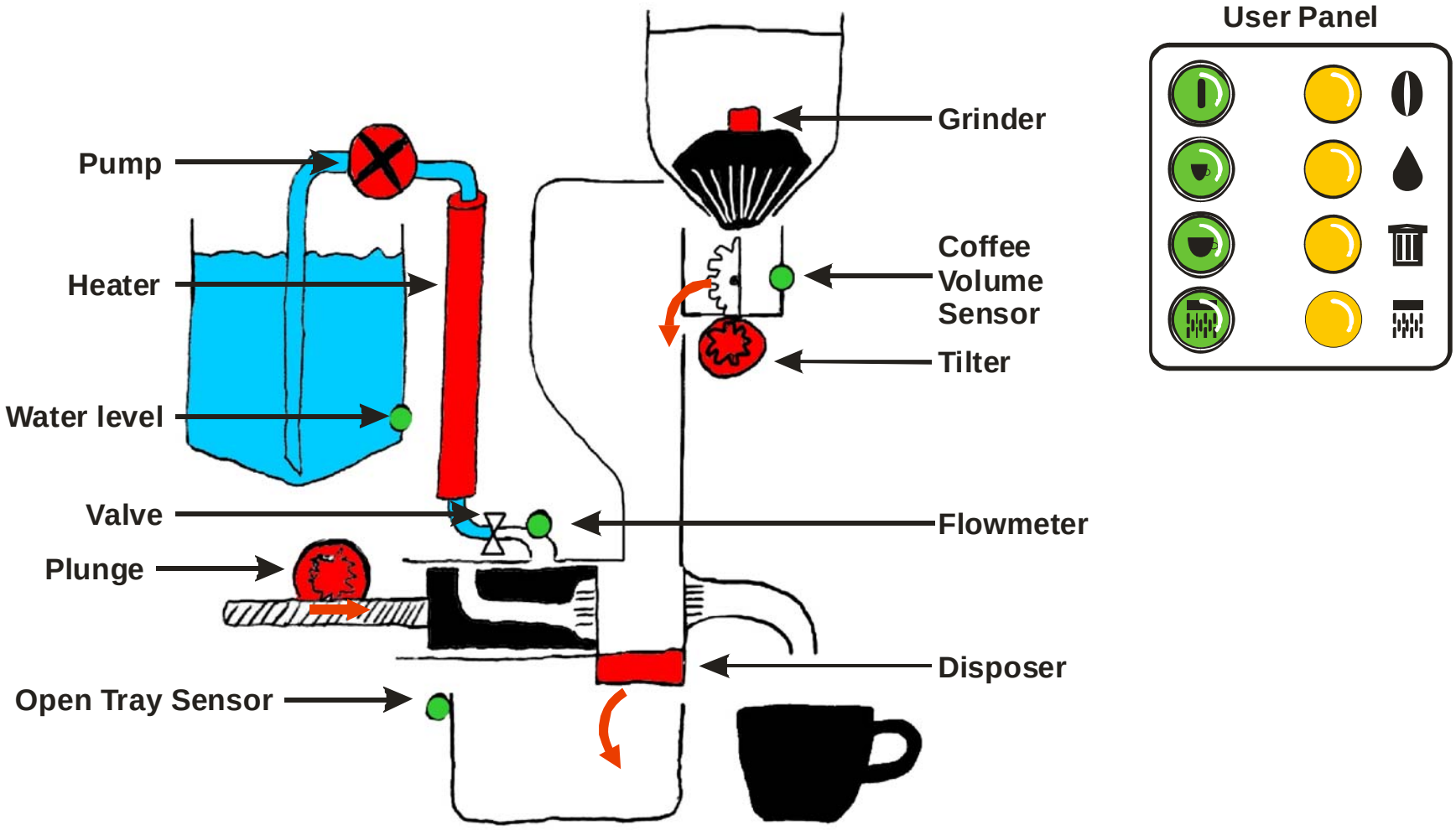
Multiple design decisions in one module
- Administration of Boxes
- Administration of Cards
- Read/Write of Cards



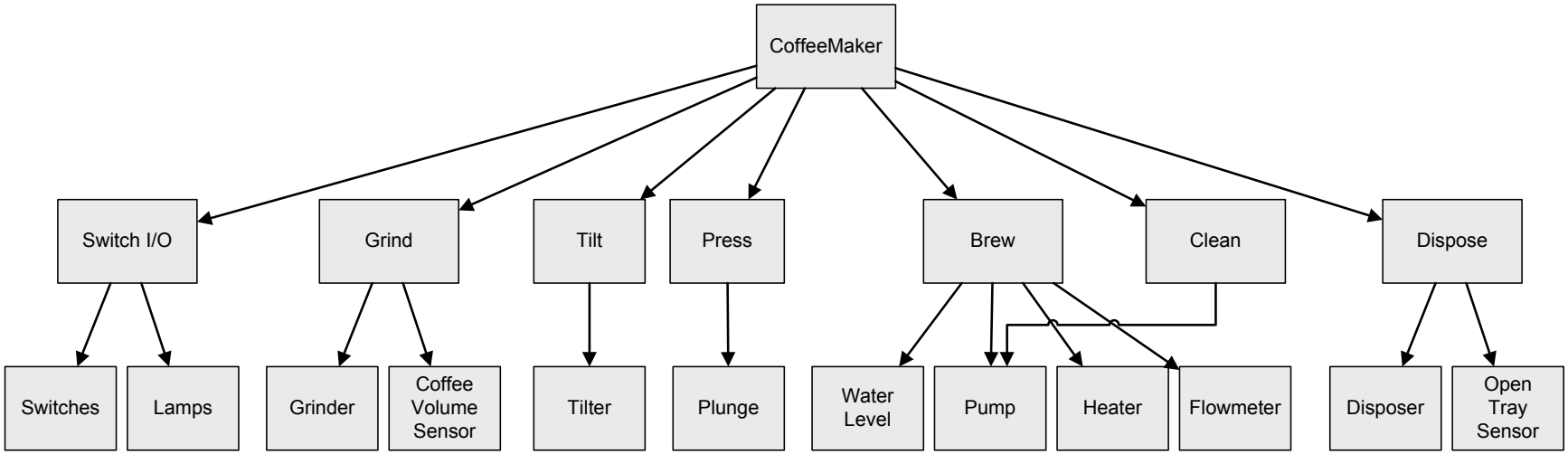
Multiple design decisions
- Entry
- Collection
- Transformation
- Representation/Transformation
- Sorting

Genericity realized by inheritance

A Simple Control System: Coffee Maker

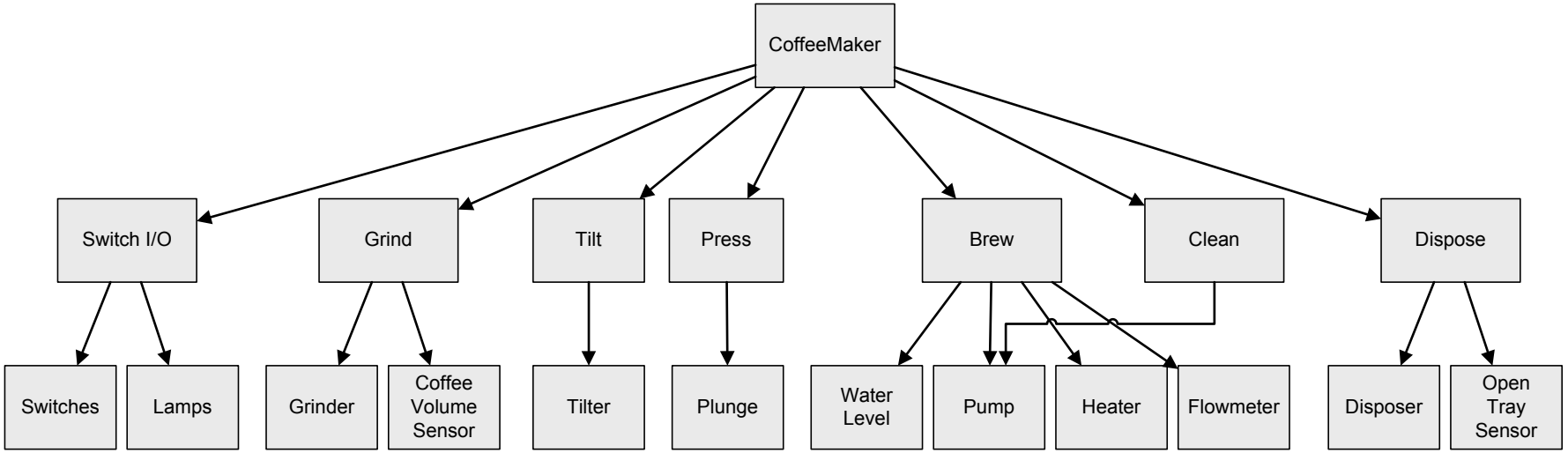


Coffee Maker – Without Knowledge



Coffee Maker – Errors made

- **realization functionality and not abstract functionality**
- **direct usage of underlying sensors, actuators and components (e.g. switches, pump, water level sensor, etc.)**
- **mixed up standard and special functions (e.g. startup/shutdown, error and exception handling)**



Summary

- Example Systems:
  - Small Batch System: functional decomposition
  - Small Interactive Problem: functional decomposition, OO
  - Small Embedded System

- Characterization:
  - Functional decomposition
    - ⇨ Tree structure, partially ordered
    - ⇨ Global data flow

    } 1:1 mapping from functional requirements
  - OO structure
    - ⇨ One inheritance structure, again a tree

    } 1:1 mapping from application entities

- Errors:
  - Functional decomposition:
    - ⇨ Data components missing
    - ⇨ Data changes cause severe system changes
    - ⇨ Requirements changes also cause severe changes
    - ⇨ Functionality as given: no abstraction
    - ⇨ Devices / Senses included: no abstractions
  - OO decomposition:
    - ⇨ Components with multiple decisions
    - ⇨ Layering: classification, not use
    - ⇨ Inheritance misunderstood
    - ⇨ Requirements changes cause severe system changes

  } these design errors occur in practice