technische universität
dortmund

fakultät für informatik
informatik 12

Embedded System Design:

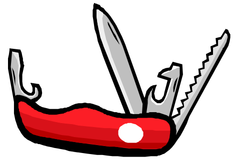
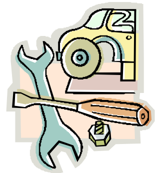
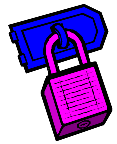
Embedded Systems Foundations of Cyber-Physical Systems

Peter Marwedel
TU Dortmund,
Informatik 12

2013年 10 月 09 日

Common characteristics

Dependability

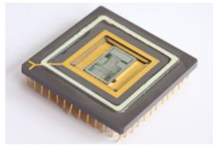
- CPS/ES must be **dependable**, 
  - **Reliability** $\boldsymbol{R(t)}$ **=** probability of system working correctly provided that is was working at t=0 
  - **Maintainability** $\boldsymbol{M(d)}$ **=** probability of system working correctly d time units after error occurred. 
  - **Availability** $\boldsymbol{A(t)}$: probability of system working at time t
  - **Safety**: no harm to be caused 
  - **Security**: confidential and authentic communication 

Even perfectly designed systems can fail if the assumptions about the workload and possible errors turn out to be wrong.

Making the system dependable must not be an after-thought, it must be considered from the very beginning

Efficiency

- CPS & ES must be **efficient**
  - Code-size efficient
    (especially for systems on a chip)



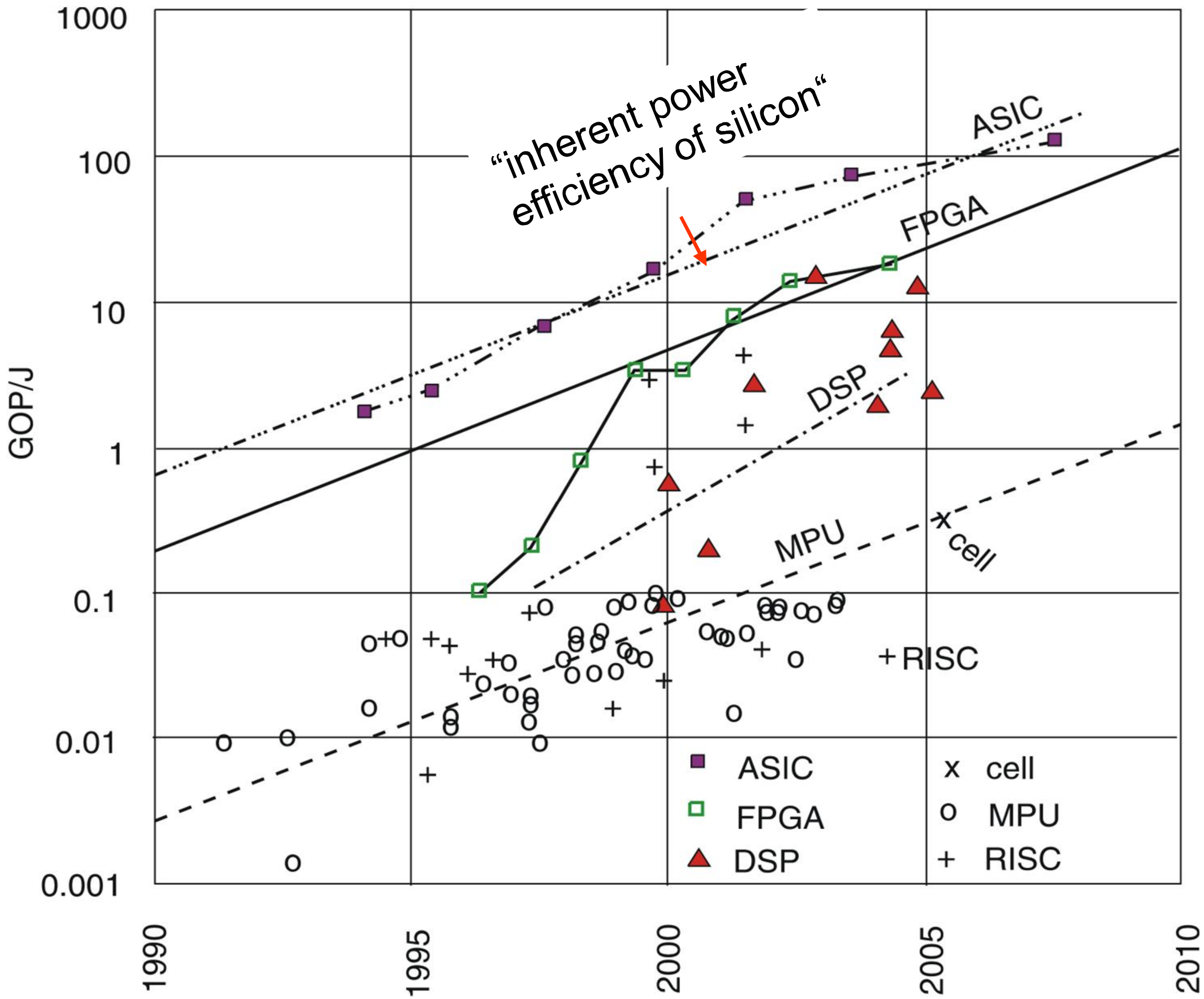
  - Run-time efficient

  - Weight efficient

  - Cost efficient

  - Energy efficient

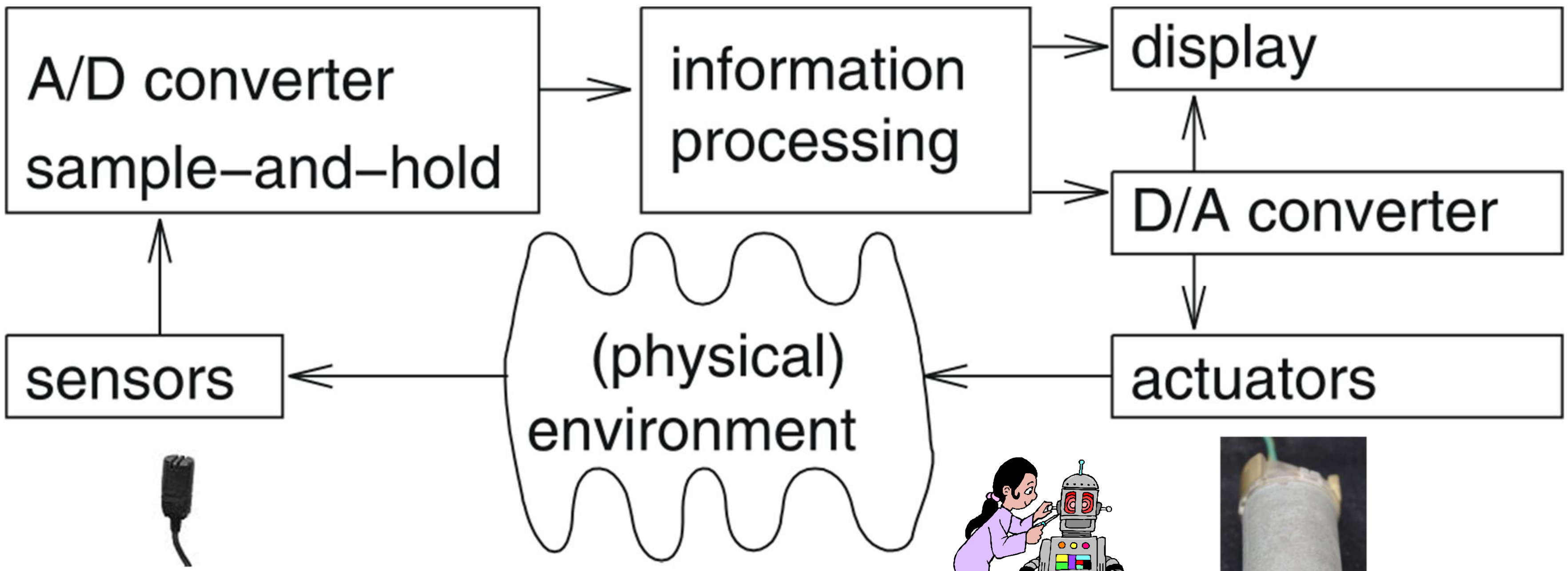
Importance of Energy Efficiency

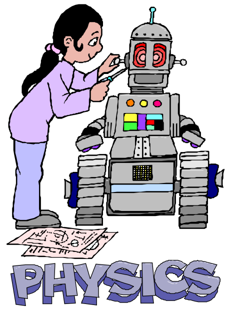


Efficient software design needed, otherwise, the price for software flexibility cannot be paid.

CPS & ES Hardware

CPS & ES hardware is frequently used in a loop
(***“hardware in a loop“***):

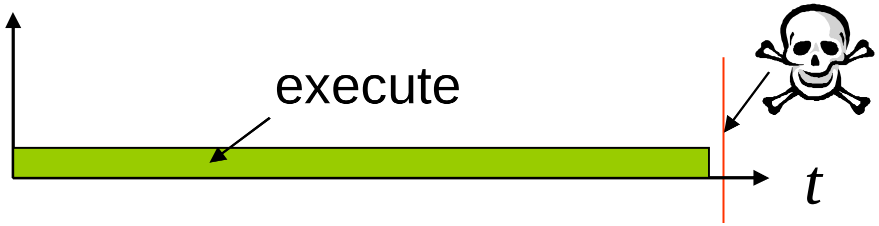


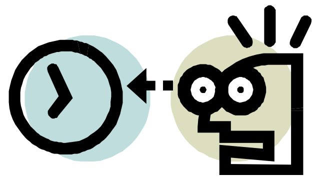


Cyber-physical systems (!)

Real-time constraints

- CPS must meet **real-time constraints**
  - A real-time system must react to stimuli from the controlled object (or the operator) within the time interval **dictated** by the environment.





  - **"A real-time constraint is called hard, if not meeting that constraint could result in a catastrophe"** [Kopetz, 1997].
  - All other time-constraints are called **soft**.
  - A guaranteed system response has to be explained without statistical arguments [Kopetz, 1997].

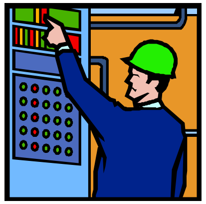
Real-Time Systems & CPS

CPS, ES and Real-Time Systems synonymous?

- For some embedded systems, real-time behavior is less important (smart phones)
- For CPS, real-time behavior is essential, hence RTS ≅ CPS
- CPS models also include a model of the physical system

Reactive & hybrid systems

- Typically, CPS are **reactive systems**:
  “**A reactive system is one which is in continual interaction with is environment and executes at a pace determined by that environment**“
  [Bergé, 1995]

  Behavior depends on input **and current state**.
  ☞ automata model appropriate,
  model of computable functions inappropriate.

- **Hybrid systems**
  (analog + digital parts).

Dedicated systems

- **Dedicated** towards a certain **application**
  Knowledge about behavior at design time can be used to minimize resources and to maximize robustness
- **Dedicated user interface**
  (no mouse, keyboard and screen)
- Situation is slowly changing here: systems become less dedicated

Security

- Defending against
  - Cyber crime („Annual U.S. Cybercrime Costs Estimated at $100 Billion; …[Wall Street Journal, 22.7.2013])
  - Cyber attacks (☞ Stuxnet)
  - Cyber terrorism
  - Cyber war (Cyber-Pearl-Harbor [Spiegel Online, 13.5.2013])

- Connectivity increases threats
  - entire production chains can be affected
  - local islands provide some encapsulation, but contradict idea of global connectedness

Dynamics

Frequent change of environment

Underrepresented in teaching

- CPS & ES are **underrepresented in teaching** and public discussions:
  "*Embedded chips aren't hyped in TV and magazine ads ...*" [Mary Ryan, EEDesign, 1995]

Not every CPS & ES has all of the above characteristics.

Def.: **Information processing systems having most of the above characteristics are called embedded systems.**

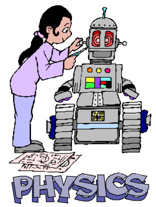
Course on embedded systems foundations of CPS makes sense because of the number of common characteristics.

Characteristics lead to corresponding challenges

- Dependability

- Efficiency
    - In particular: Energy efficiency

- Hardware properties, physical environment



- Meeting real time requirements

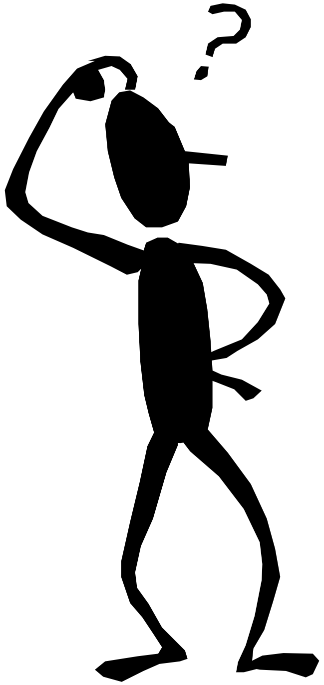
- ….

Challenges for implementation in hardware

- Early embedded systems frequently implemented in hardware (boards)
- Mask cost for specialized application specific integrated circuits (ASICs) becomes very expensive (M$ range, technology-dependent)
- Lack of flexibility (changing standards).
- ☞Trend towards implementation in software (or possibly FPGAs, see chapter 3)

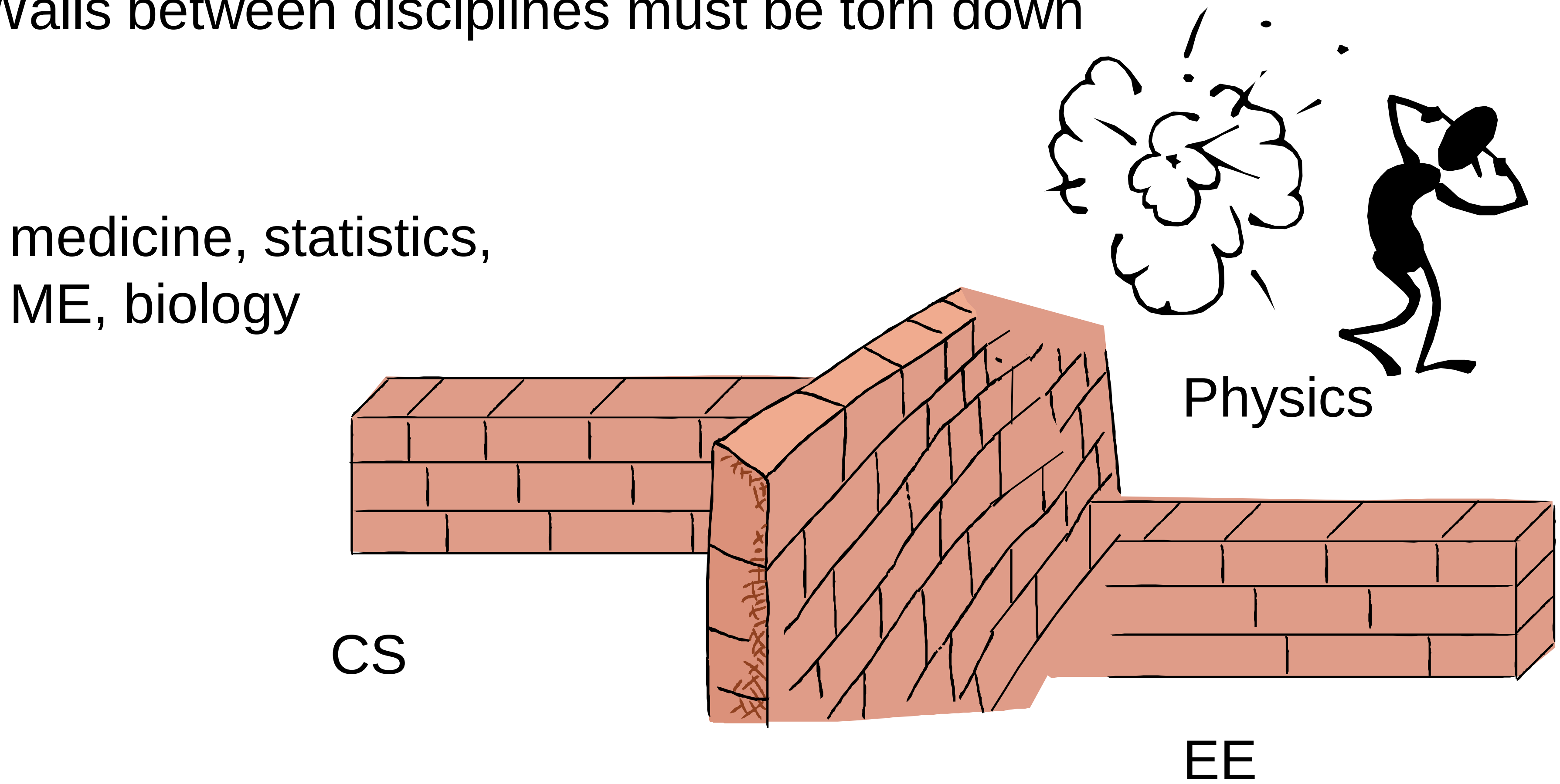
Challenges for implementation in software

If CPS/ES will be implemented mostly in software, then why don‘t we just use what software engineers have come up with?



It is not sufficient to consider CPS/ES as a special case of SW engineering

Knowledge from many areas must be available,
Walls between disciplines must be torn down



Challenges for CPS/ES Software

- Dynamic environments
- Capture the required behaviour!
- Validate specifications
- Efficient translation of specifications into implementations!
- How can we check that we meet real-time constraints?
- How do we validate embedded real-time software? (large volumes of data, testing may be safety-critical)





© Graphics: P. Marwedel, 2011

Software complexity is a challenge

Software in a TV set

- Source 1*:

Year	Size
1965	0
1979	1 kB
1990	64 kB
2000	2 MB

- Source 2°: 10x per 6-7 years

Year	Size
1986	10 KB
1992	100 kB
1998	1 MB
2008	15 MB

- ☞ Exponential increase in software complexity
- *... > 70% of the development cost for complex systems such as automotive electronics and communication systems are due to software development* [A. Sangiovanni-Vincentelli, 1999]

* Rob van Ommering, COPA Tutorial, as cited by: Gerrit Müller: Opportunities and challenges in embedded systems, *Eindhoven Embedded Systems Institute*, 2004

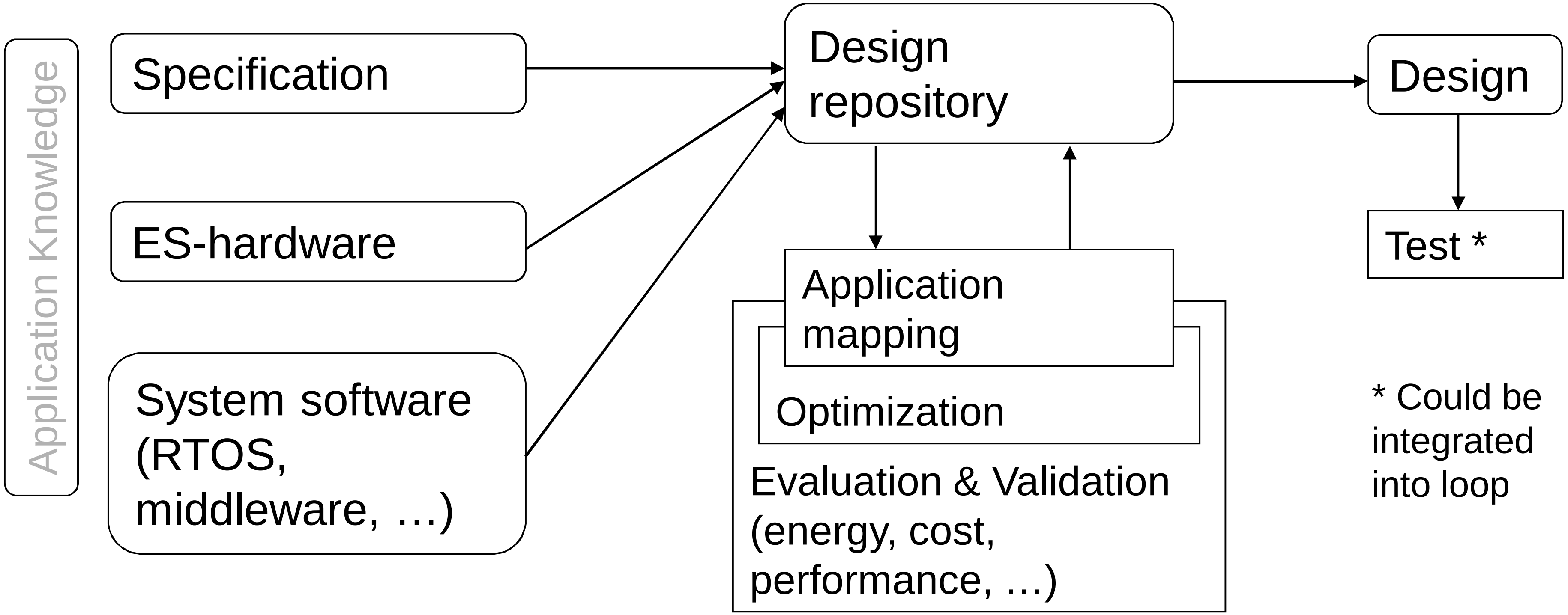
° R. Kommeren, P. Parviainen: Philips experiences in global distributed software development, *Empir Software Eng*. (2007) 12:647-660

technische universität
dortmund

fakultät für informatik
informatik 12

Design flows

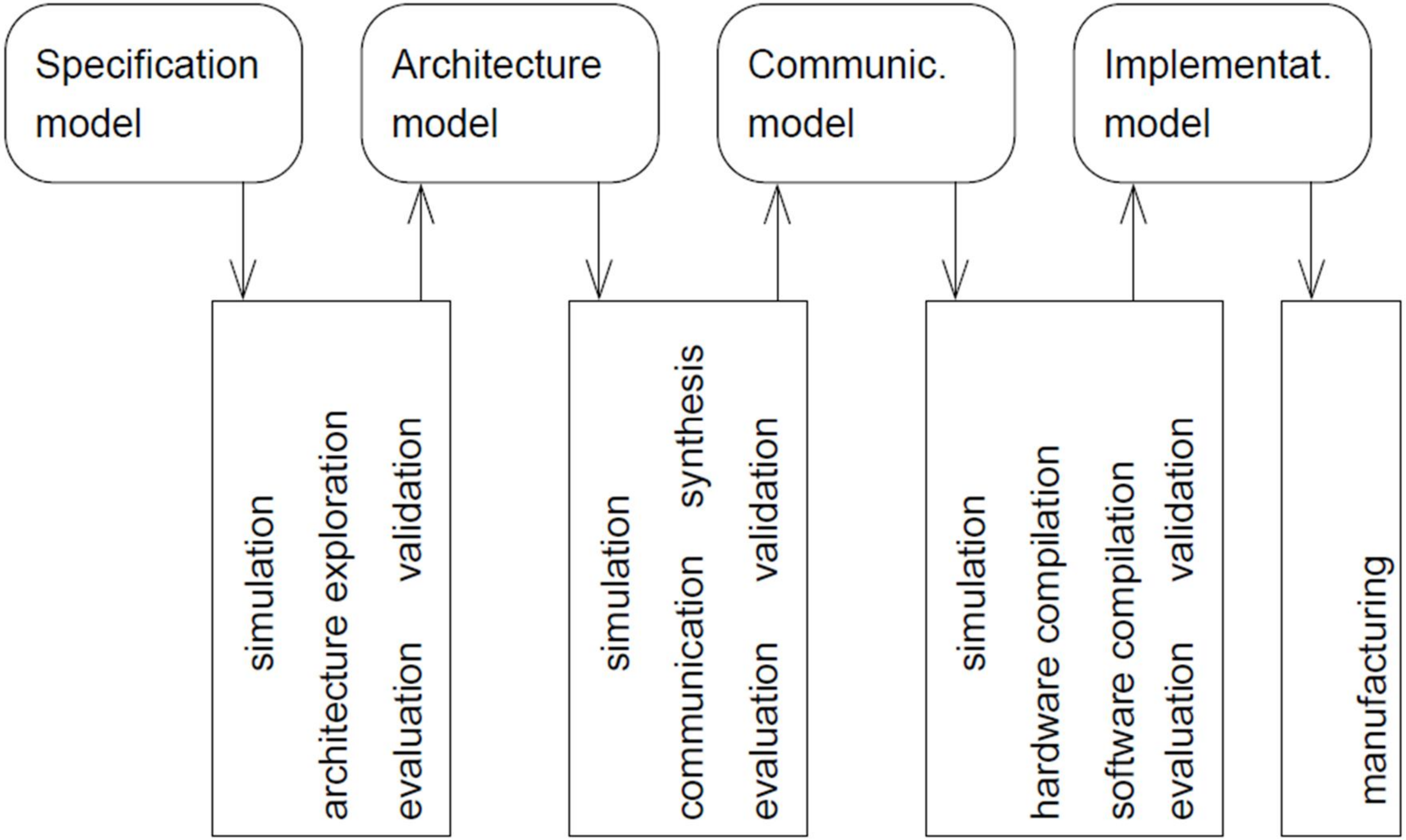
Hypothetical design flow



Generic loop: tool chains differ in the number and type of iterations

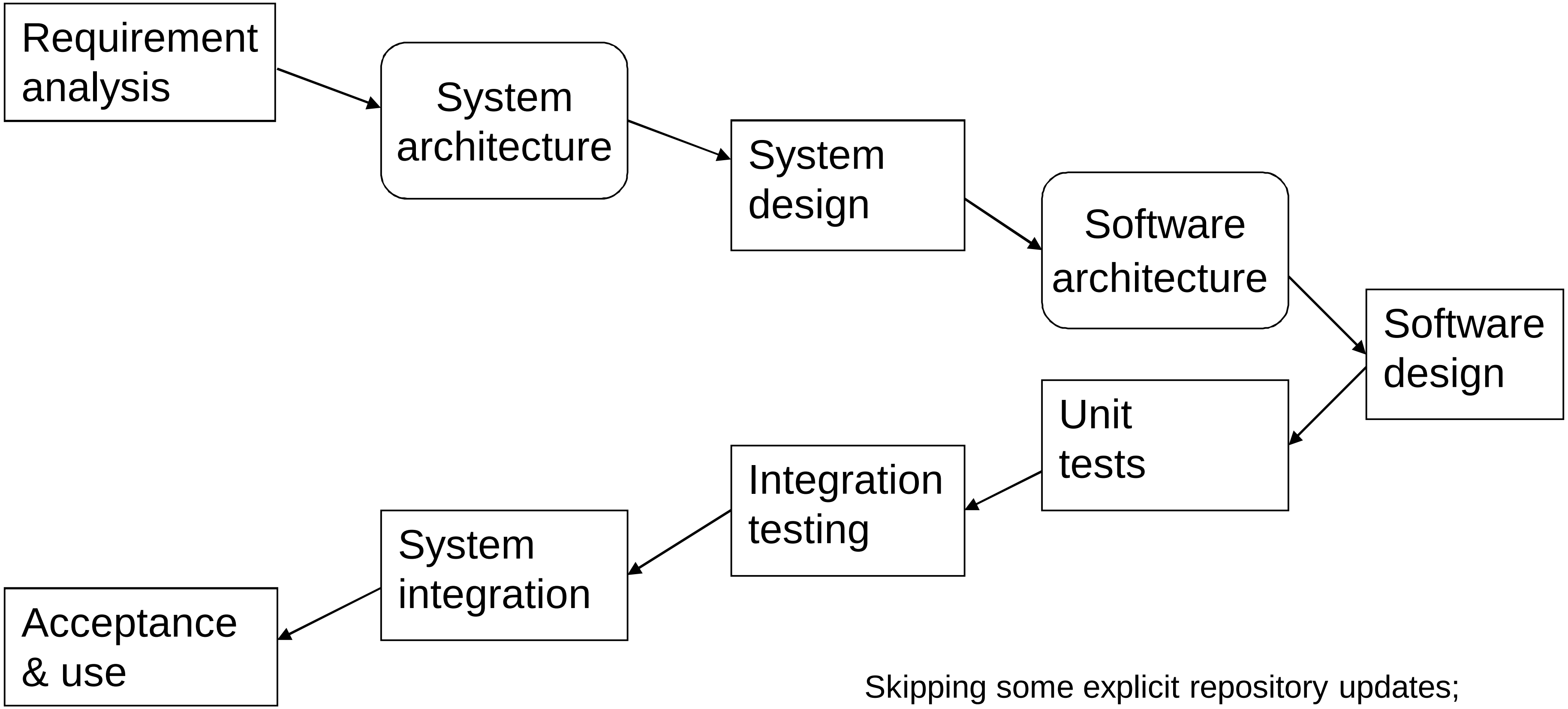
Iterative design (1): - After unrolling loop -

Example:
SpecC
tools



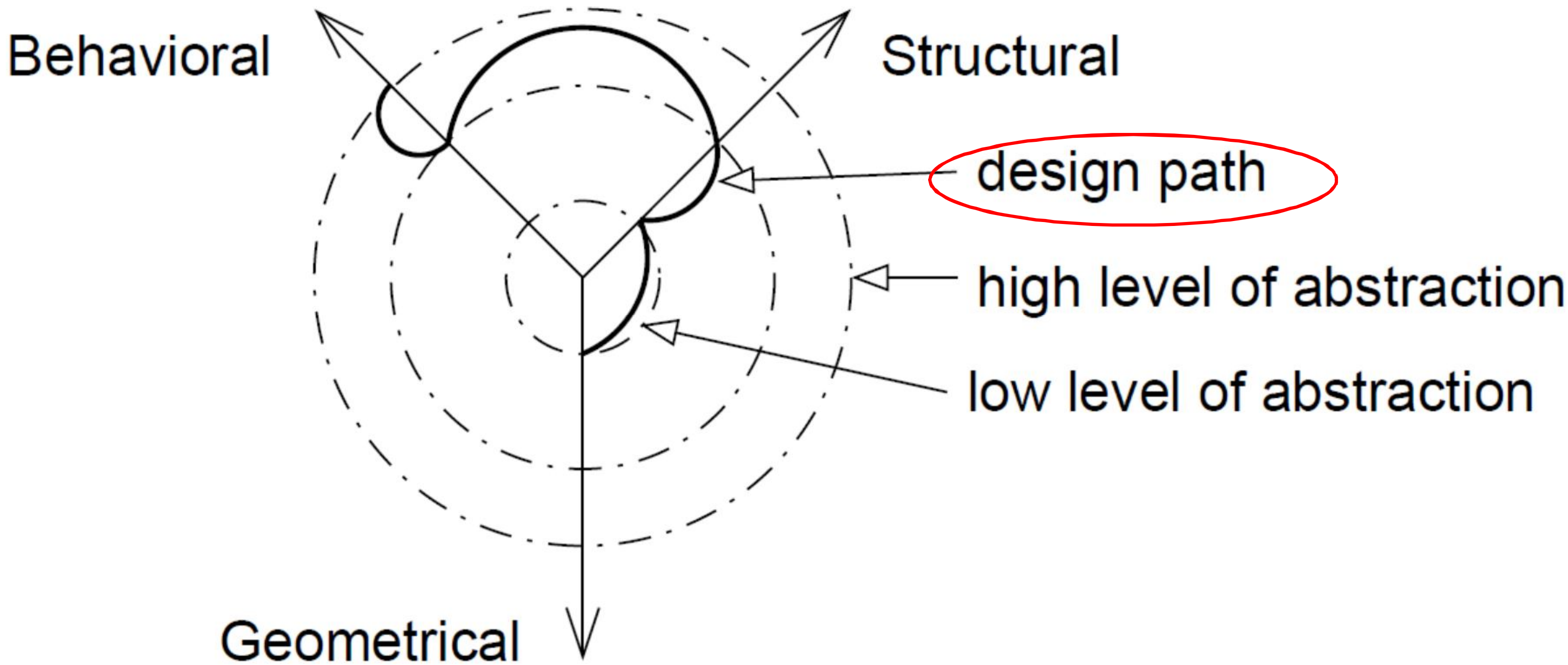
Iterative design (2): - After unrolling loop -

Example: V-model



Skipping some explicit repository updates;
very late integration, problems may be missed ..

Iterative design (3): - Gajski's Y-chart -



Summary

- Common characteristics
- Challenges (resulting from common characteristics)
- Design Flows