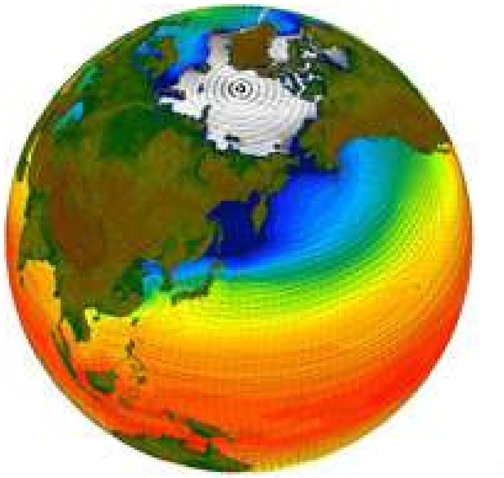
The Two Sides of Design and Implementation

Bergamo, October 29th 2019

Henri Stephanou
Paris-I University

How do programs relate to reality?



Scientific computing

- *Has impact only on the mind of the scientist*
- *Yields knowledge on the natural and socio-economic world*

Business computing

- *Has impact on the mind of people acting in the world*
- *Has information on a subset of human affairs*

Automation software

- *Has causal impact on a mechanical machine*
- *Yields very narrow information, on its immediate environment only*

Plan

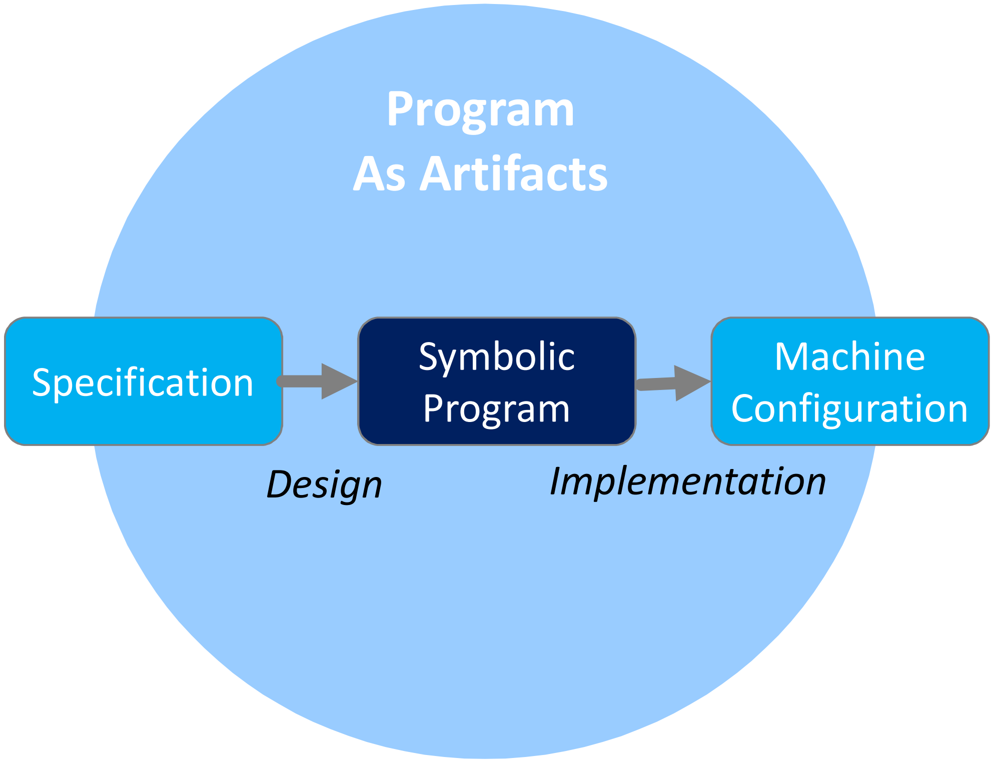
The classical view: « Logical firewalls »

Case studies challenging the classical view

Michael Jackson's view and its limits

Final proposal

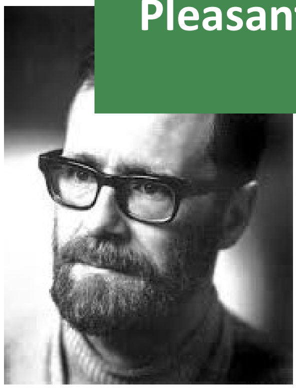
Program artifacts



« Specification provides the function, a symbolic program is taken as the structural description, and the physical process is generated by the implementation. […]

We shall call these ontological bundles program artifacts. » (Turner 2018, p. 52)

Logical firewalls



Pleasantness problem: non-relevance of the context of use

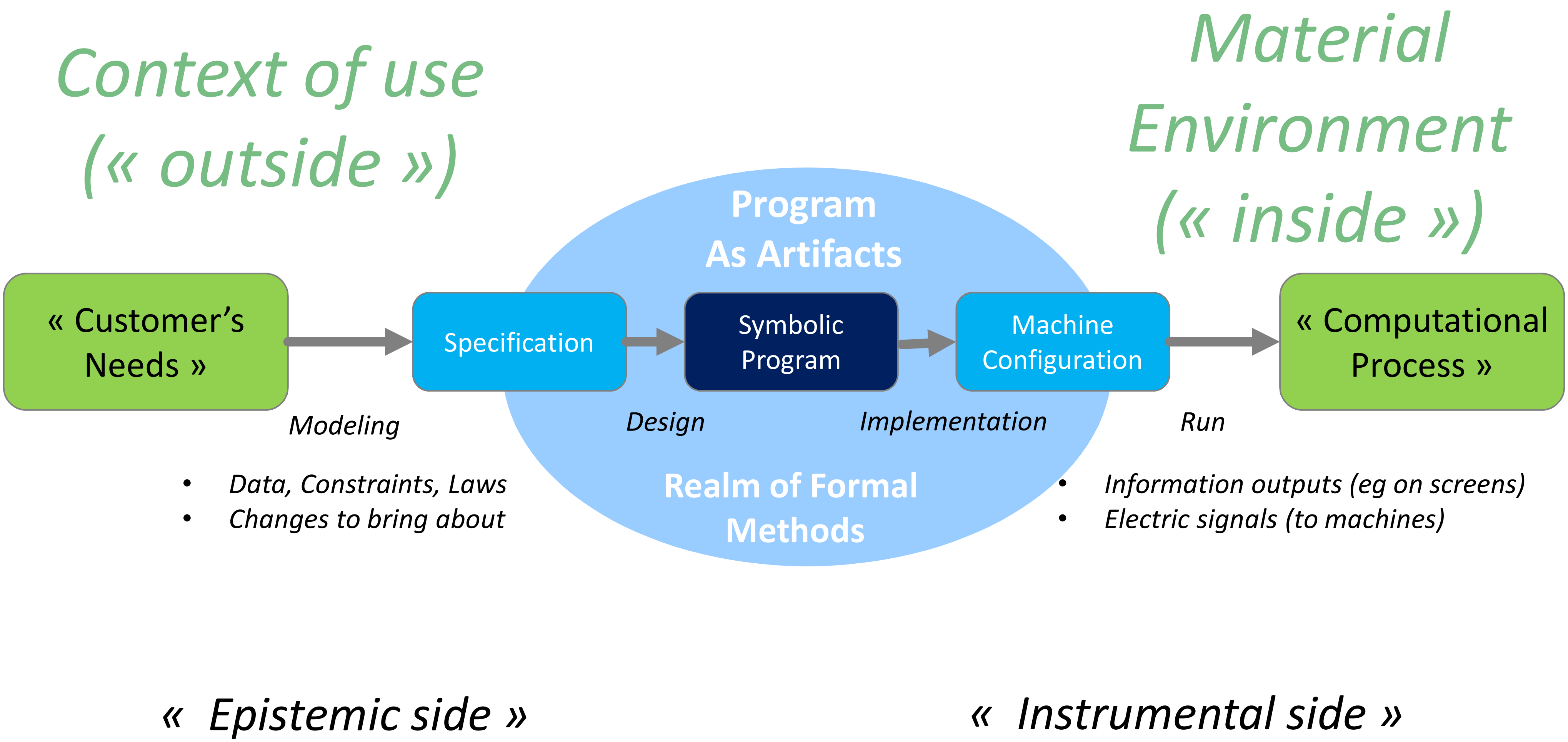
*"The role of a formal functional specification is simply to act as a logical firewall between two completely different concerns, known under the names of "the pleasantness problem" and "the correctness problem".
The pleasantness problem concerns the question whether a system meeting such-and-such a formal functional specification would satisfy our needs, meet our expectations and fulfil our hopes. The correctness problem concerns the question whether a given design meets such-and-such a formal functional specification." (Dijkstra EWD 952)*

Materiality argument: non-relevance of the material environment

« When the correctness of a program, its compiler, and the hardware of the computer have all been established with mathematical certainty, it will be possible to place great reliance on the results of the program, and predict their properties with a confidence limited only by the reliability of the electronics » (Hoare 1969)

«In [simple cases], it might be argued that the abstract machine is the target machine. But […] an abstract machine no more qualifies as a machine than an artificial flower qualifies as a flower. Compilers, interpreters, processors and the like are properly characterized as physical things, i.e., as systems in space/time for which causal relations obtain. » (Fetzer, 1988)

The broader classical view



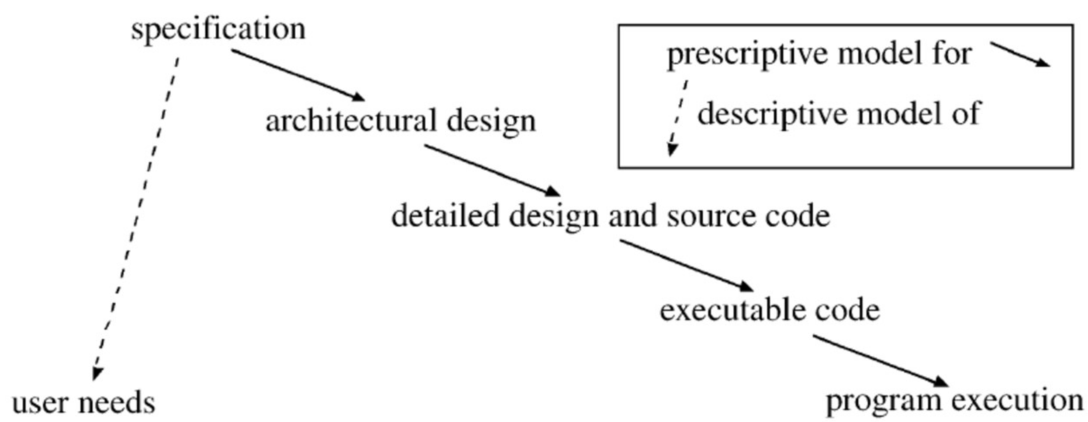
A slippery distinction: descriptive vs. prescriptive

"A model can mirror an existing original (like a photograph), or it can be used as a specification of something to be created (like a construction plan). In the former case, we call it a descriptive model; in the latter case, we call it prescriptive." (Ludewig, 2003, p.8)

Examples:
- *documentation (descriptive)*
- *instructions (prescriptive)*
- *prototypes (descr. then prescr.)*
- *games (descriptive)*
- *formal models (descriptive)*

Software engineering models (e.g. use cases, flow or class diagrams, design patterns) are mostly prescriptive: explain how to build the software

« The requirements specification is double-sided, because it describes the user's needs, and it prescribes the product to be developed. It is this double role that makes the specification the most important software component"



« user manual and test data are descriptive models of the specification; they can replace it for certain purposes»

Three facts requestioning this picture

1. **Run-time considerations influence software's design**

2. **Design and implementation often mix up when external systems are involved**

3. **Some programs require a strong change in the users' behavior; implementation also happens on the users' side!**

Example n°1:
Massively Parallel Computations: The machine invites herself back (1)

Order matters !

- Associative property does not hold

  a + (b + c) <> (a + b) + c

Example with

a=11.0000b b=0.000011b c=0.000001b

- Each value can be represented in the computer with 6 bits mantissa without problems
- What about the result of a+b+c ?

```
a         = 11,0000        (a+b)   = 11,0000 11
 +(b+c)=   0,0001 00            +c=   0,0000 01
===================      ===================
a+(b+c)= 11,0001          (a+b)+c= 11,0000
```

With every mathematical operation in a computer you will be faced with that problem !

With a big high performance parallel computer you subdivide your task into small portions, have them computed on different processors, and collect and merge the individual results

Approximately 100,000,000,000,000 operations/sec

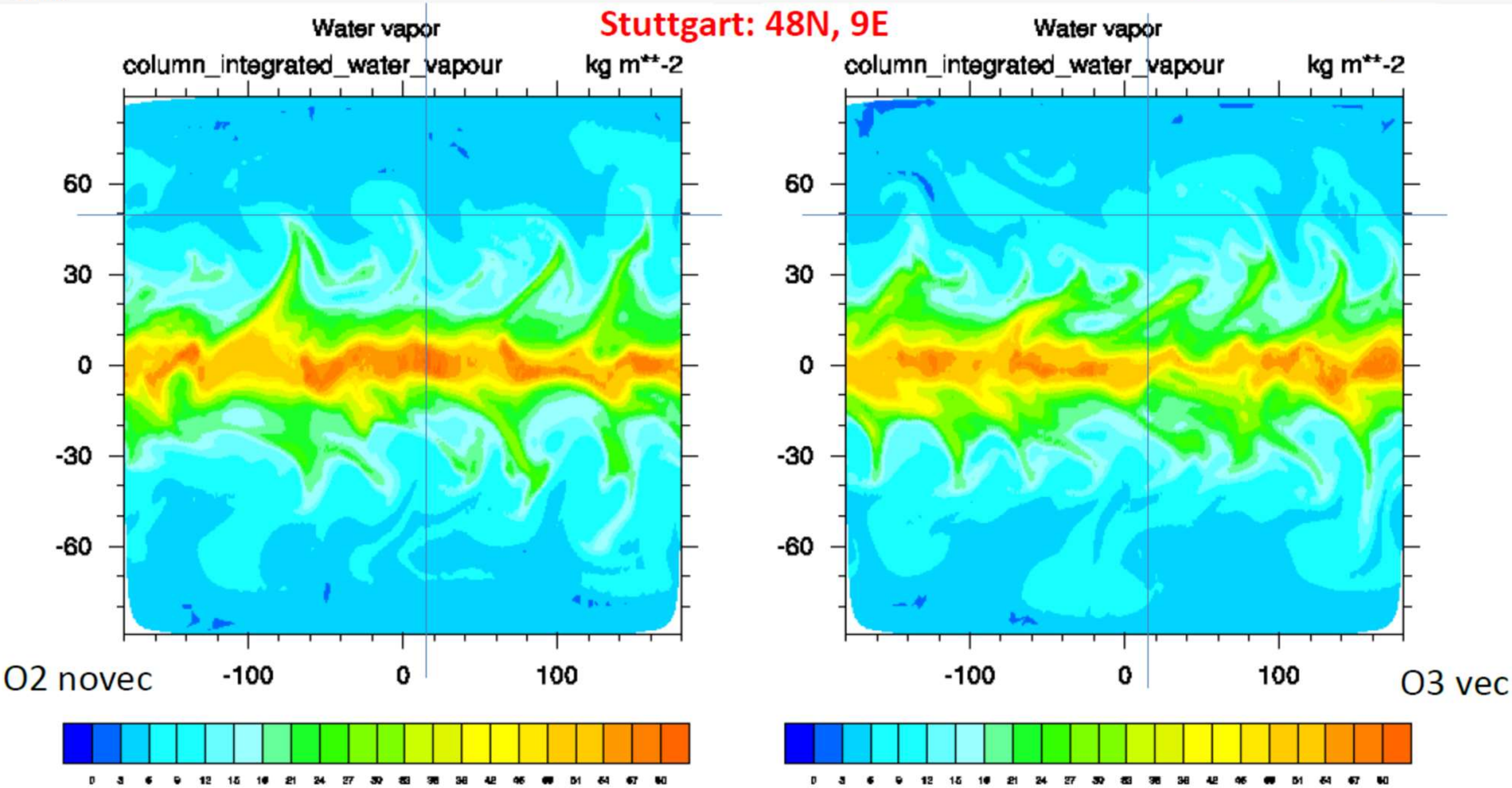
100 trillion

with 100,000 processors on DKRZ machine

Source: *Ludwig, 2018 ©*

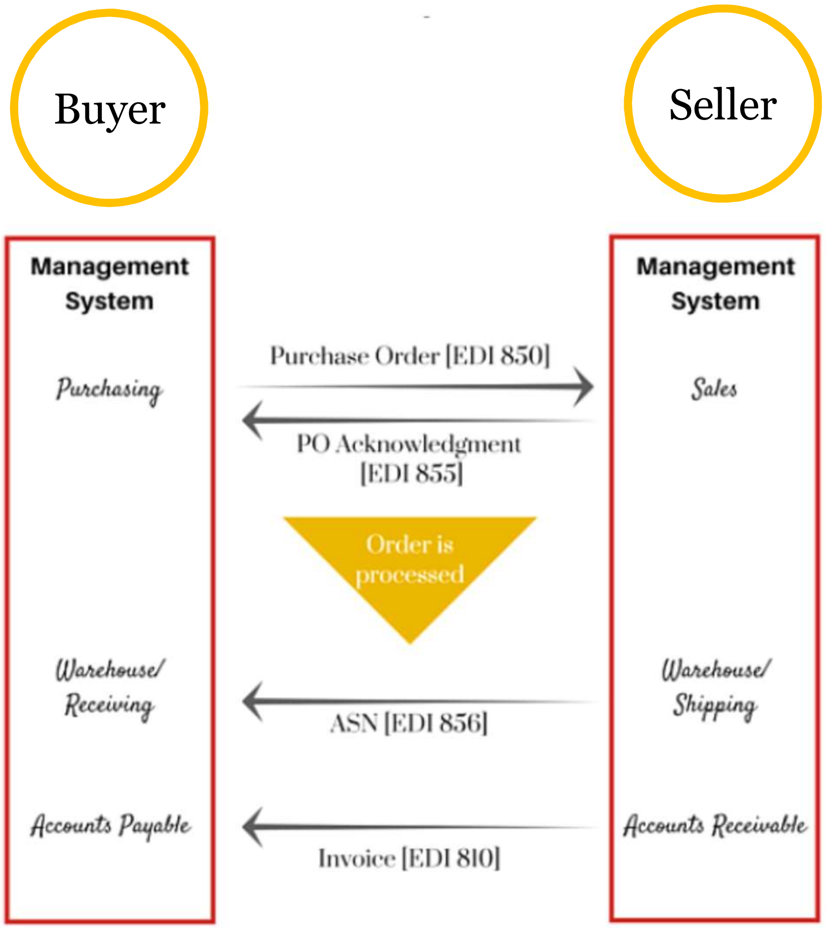
Example n°1:
Massively Parallel Computations: The machine invites herself back (2)

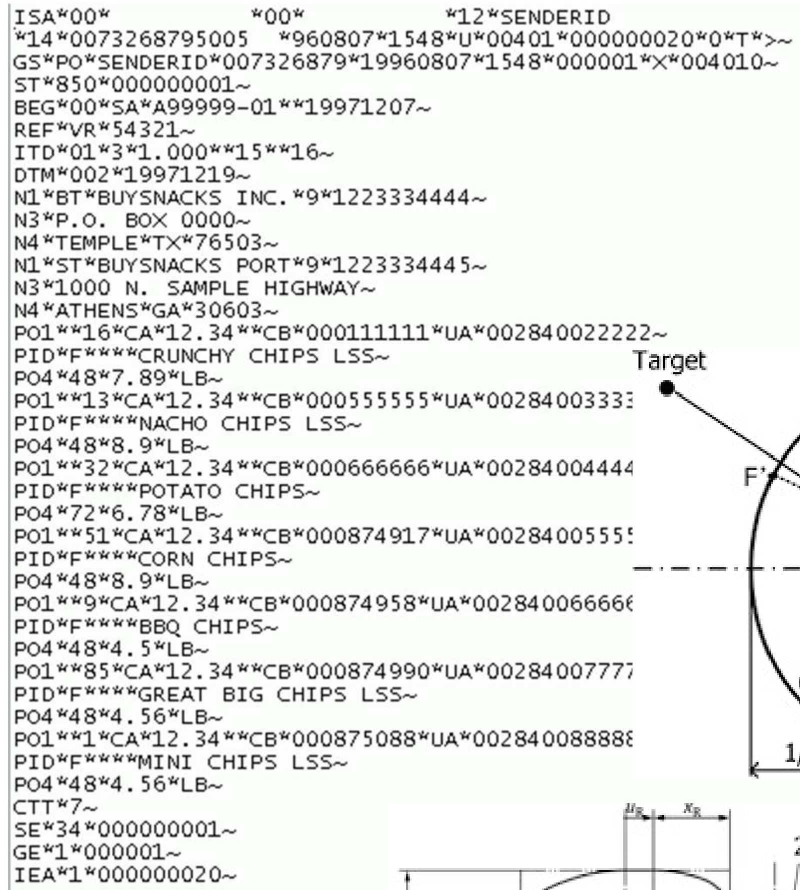
t_{sim}=100d O2: 10min30sec -> O3: 8m41sec => (+17,3%)



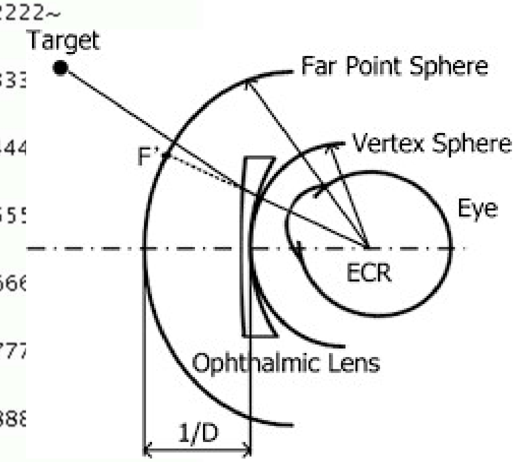
Source: *Ludwig, 2018 ©*

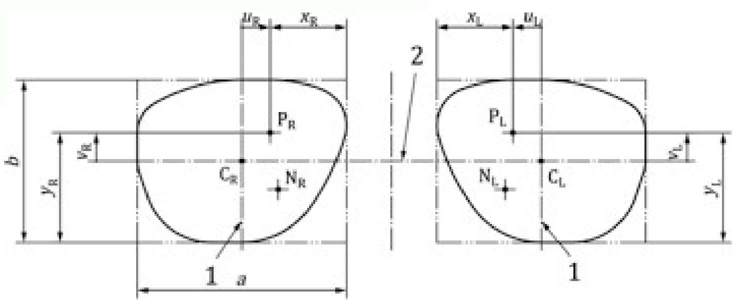
Example n°2:
The messy world of EDI (Electronic Data Interchange) (1)





```
ISA*00*          *00*          *12*SENDERID
*14*0073268795005  *960807*1548*U*00401*000000020*0*T*>~
GS*PO*SENDERID*007326879*19960807*1548*000001*X*004010~
ST*850*000000001~
BEG*00*SA*A99999-01**19971207~
REF*VR*54321~
ITD*01*3*1.000**15**16~
DTM*002*19971219~
N1*BT*BUYSNACKS INC.*9*1223334444~
N3*P.O. BOX 0000~
N4*TEMPLE*TX*76503~
N1*ST*BUYSNACKS PORT*9*1223334445~
N3*1000 N. SAMPLE HIGHWAY~
N4*ATHENS*GA*30603~
PO1**16*CA*12.34**CB*000111111*UA*002840022222~
PID*F****CRUNCHY CHIPS LSS~
PO4*48*7.89*LB~
PO1**13*CA*12.34**CB*000555555*UA*00284003333
PID*F****NACHO CHIPS LSS~
PO4*48*8.9*LB~
PO1**32*CA*12.34**CB*000666666*UA*00284004444
PID*F****POTATO CHIPS~
PO4*72*6.78*LB~
PO1**51*CA*12.34**CB*000874917*UA*00284005555
PID*F****CORN CHIPS~
PO4*48*8.9*LB~
PO1**9*CA*12.34**CB*000874958*UA*00284006666
PID*F****BBQ CHIPS~
PO4*48*4.5*LB~
PO1**85*CA*12.34**CB*000874990*UA*00284007777
PID*F****GREAT BIG CHIPS LSS~
PO4*48*4.56*LB~
PO1**1*CA*12.34**CB*000875088*UA*00284008888
PID*F****MINI CHIPS LSS~
PO4*48*4.56*LB~
CTT*7~
SE*34*000000001~
GE*1*000001~
IEA*1*000000020~
```



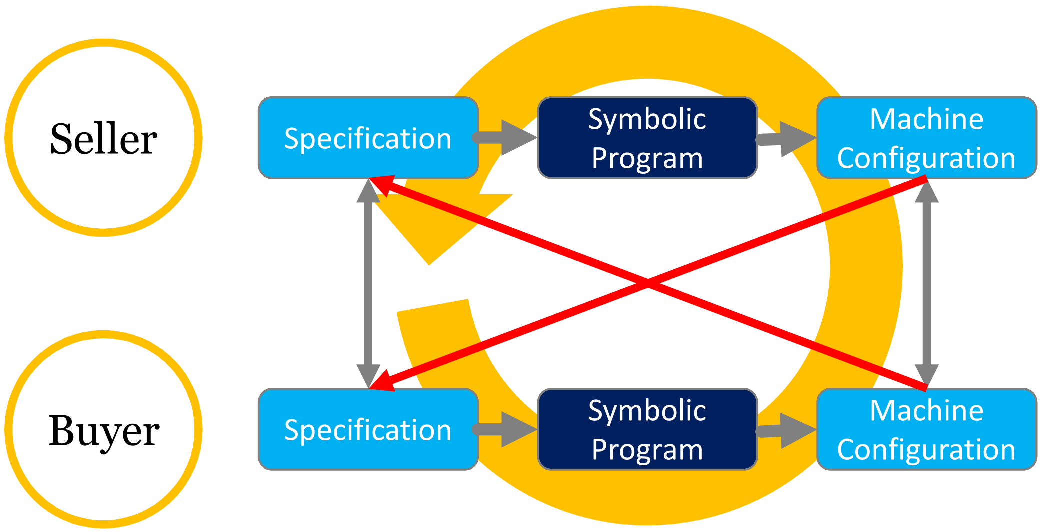
Example n°2:
The messy world of EDI (Electronic Data Interchange) (2)

An EDI project by the Book…

- Get specs of information to be exchanged
- Develop translation algorithm & software
- Agree on exchange specifications
- Implement communication channel
- Test

… The Reality

- Missing information in data referential
- Information mappings issues
- Material translation issues (special caracters, trailing caracters…)
- Compatibility of communication channels
- Volume overload
- Time-outs
- …

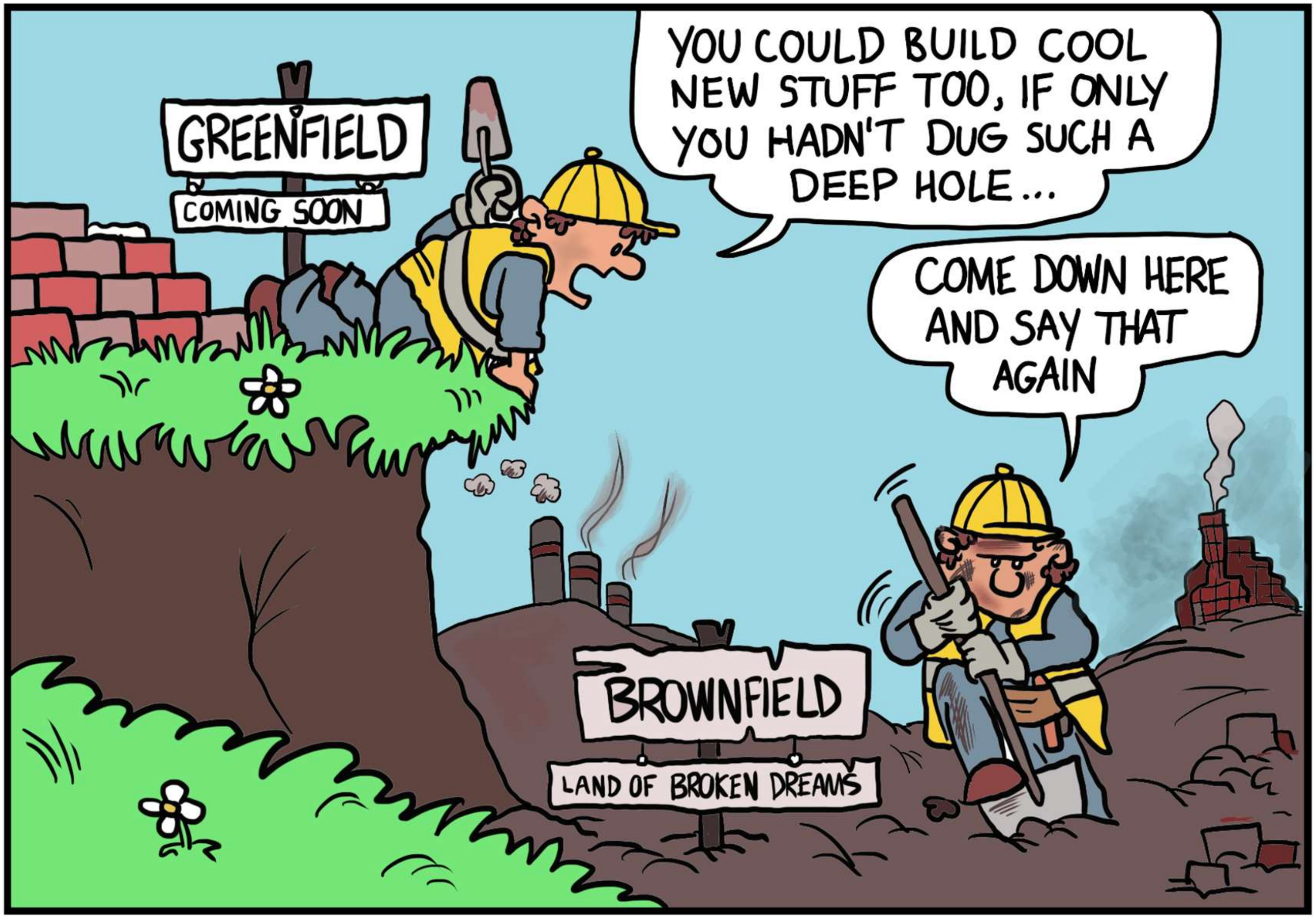


Specification is both ways: our program must adapt to the external system's specification, but may also put constraints to it!

Material considerations heavily influence specification

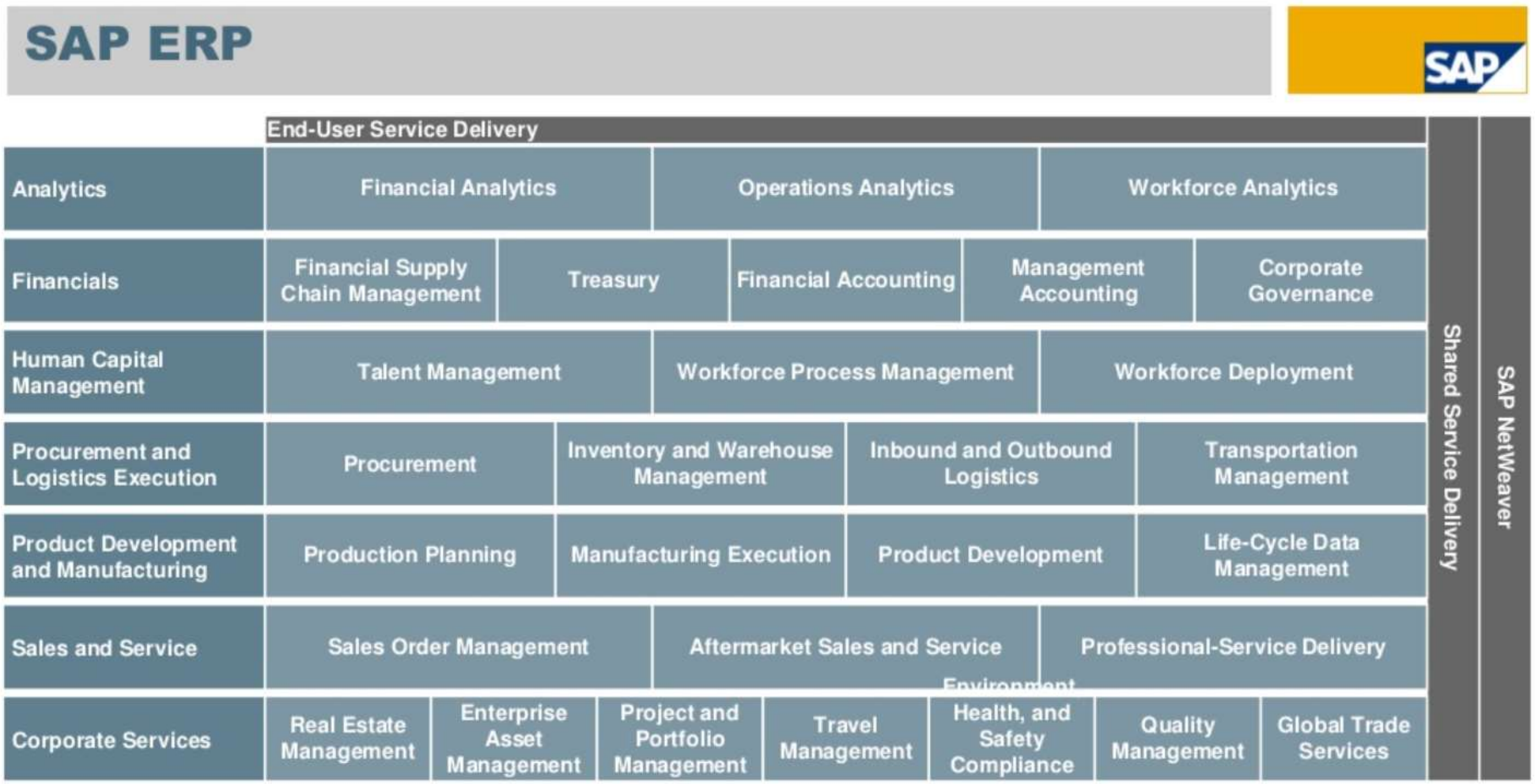
Data, exchange and communications specs and implementation deeply intertwined

Interlude: Greenfield vs. Brownfield

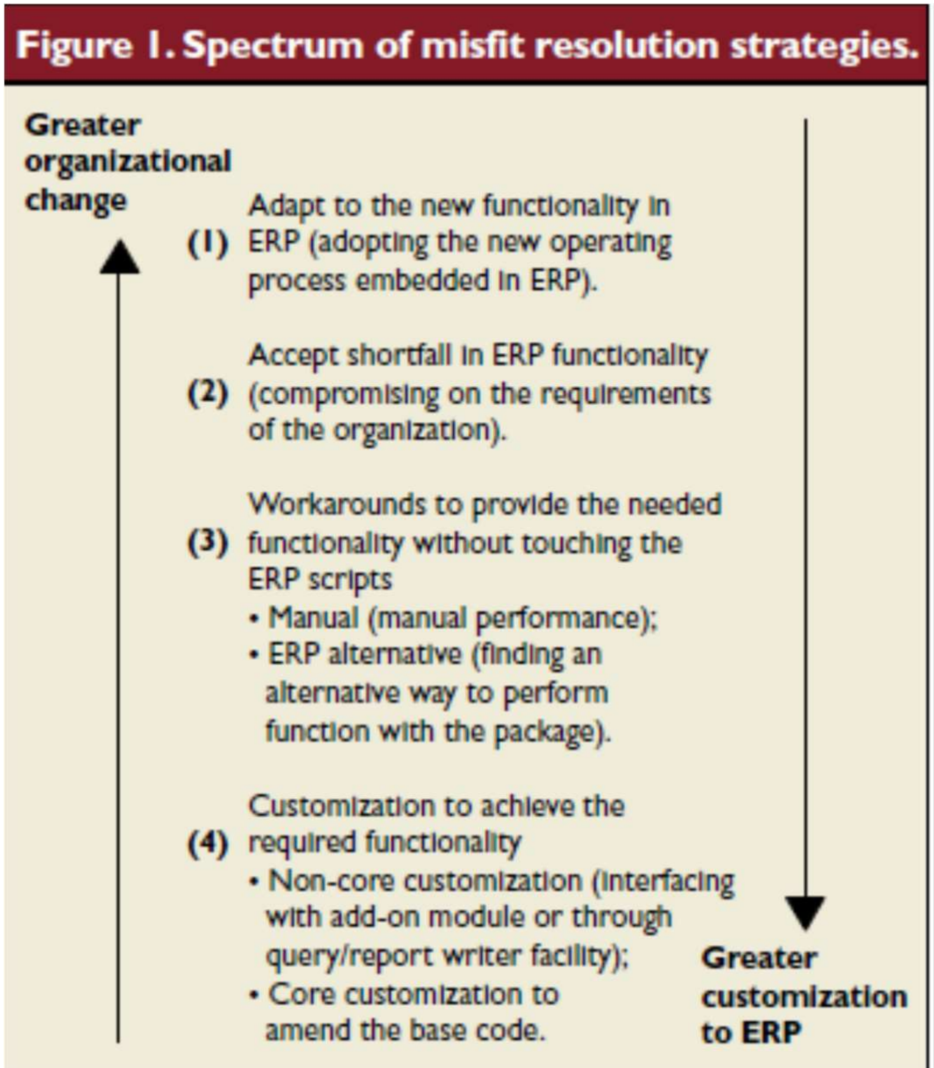


Example n°3:
ERP process reference models

Illustration: SAP reference map – global view



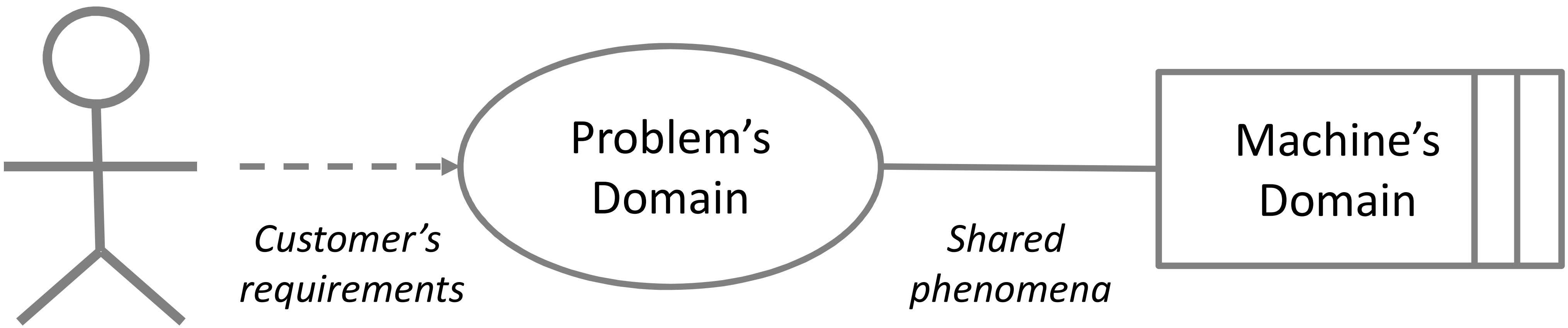
ERPs require strong change management projects on the users' side



Purchase orders (PO) and payment processes are tighly linked in the ERP referential, while they relied on manual checks previously.	Integrate and remove redundant operations and positions
ERP system has Western name syntax as first, middle, and last name. Asian staff have a difficult time understanding which part of an Indian, Malay, or Chinese name should be considered last or first name.	Workaround within the ERP: enter Asian name as Last name field, continue in First name field if name is greater than 30 characters.
ERP's patient management module does not allow the patient to pay the bill by a fixed amount every month, tracking the outstanding amount per installment etc.	Develop add-on module to ERP patient management system to handle billing and collection.

Source: *Soh et al, 2000*

Michael Jackson's Problem and Machine Domains



"[People] assumed implicitly that the phrase "software engineering" *was to be narrowly interpreted, […] that it was primarily concerned with the processes of software design, programming and testing, and with program execution.*
The alternative broader interpretation of the phrase<u>*, to mean the engineering of change in the world by devising and installing software-intensive systems*</u>*, was not seriously considered. » (Jackson 2005, 903)*

"Because problems are located in the world, problem analysis must be concerned with the world and its phenomena. We need a phenomenology that has nothing to do with programming languages or object interaction, but everything to do with the physical world. […]

It is useful to distinguish […] causal, lexical and biddable domains. All are physical domains, but demand different kinds of description and raise different development concerns […] *(Jackson 2001)*

Biddable users?

“If positive behaviour is required of a system operator or user , [she] is bidden*, or enjoined, follow the instructions.[…].*

The extent to which correct behaviour, in this sense, can be relied on varies over a wide spectrum. The pilot of a plane or the driver of a train can be relied on to behave correctly almo always. […]

By contrast, the user of an ATM can not be expected to adhere to an instruction manual. The appropriate domain properties description ***<u>must accommodate every behaviour that is physically possible</u>***.» *(Jackson 2001)*

The Washington Post

How a fish tank helped hack a casino

Unknown hackers recently attempted to steal from an unspecified North American casino by hijacking a high-tech smart fish tank.

The internet connected fish tank, which featured advanced sensors that "automatically regulate temperature, salinity, and feeding schedules," was configured to an individual VPN, to ensure that its communications network and data were safe from the hands of malicious entities. However, despite the security measures the casino took to secure the smart fish tank, hackers still managed to gather data.

Security researchers at Darktrace said that the hackers managed to steal some data and send it to a device in Finland before the attack was stopped (…)   (International Business Times 2017)

The messiness of requirements

*1) **Objectives** [such as] "to reduce by 20% waiting time at the counter" [...] are expressed, in the majority of cases, by the client. The verb implied is the verb **want** [...]: we (the company) want to decrease the waiting time [...]*

*2) **the use cases** or scenarios correspond to the needs of users and [...], for example "register a new customer". The implied phrase is **need**, expressed in the first person: "I need to register any new client. "*

*3) **The rules** are regulatory requirements or business rules (also called business rules). The verb implied or explicit is **required** [...] For example: "A withdrawal of cash can only be made if the account is positive. »[...]*

*4) **Functional requirements** are the heart and often the most important part of a specification. They express a required behavior on the part of the system. They derive from the previous categories. For example: "If cash withdrawal is not allowed, the system sends a message to the customer. "*

*5) **Quality requirements**, also called non-functional requirements, although they are only part of them. They express themselves in the form of an **adjective (fast, easy** ...)*

*6) **Interface requirements** that express the need for communication between the system under study and the outside world: hardware, software and people.*

*7) **Technical constraints**, such as the use of a particular system or language, or specific technologies, such as a communication or security protocol.*

*8) **Data formats** requirements such as postcode, country code, etc.*

*9) **Other information or requirements**, e.g. legacy system description, constraints on delays, costs, etc.*

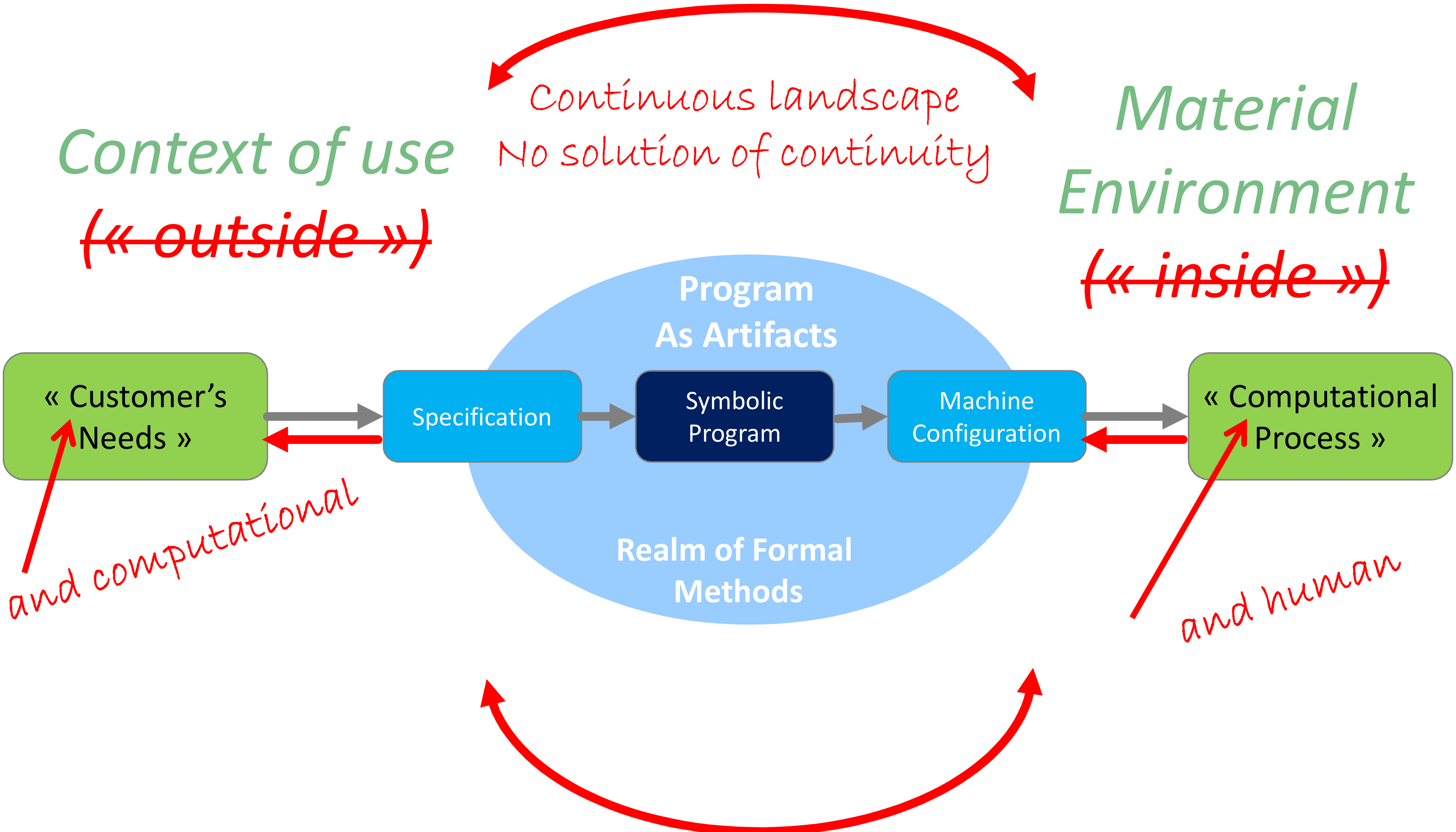
(Constantinidis 2015, p.108)

The doom loop of requirements

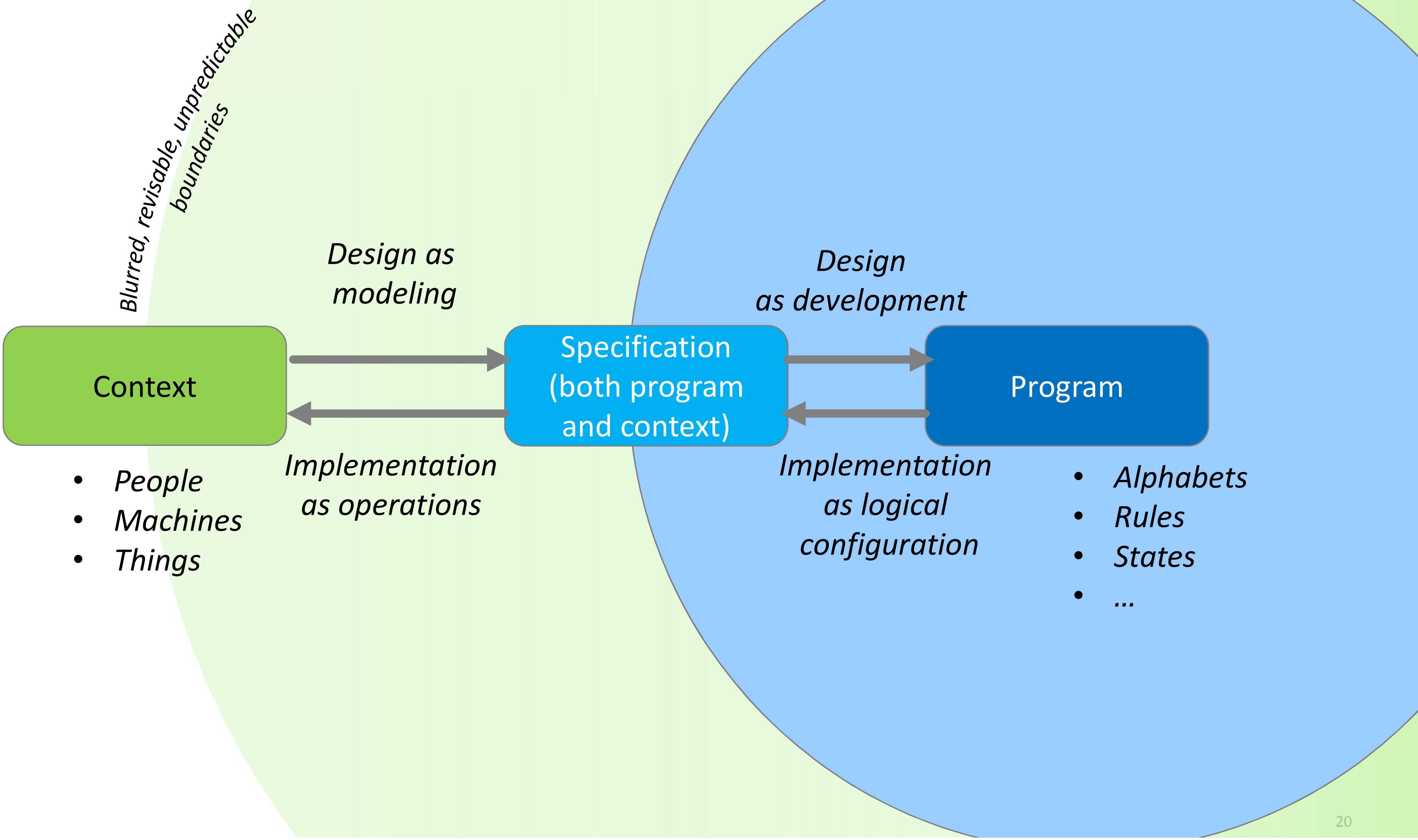
"Representations (descriptions, determinations of many kinds) of 'what the machine is' take their sense from descriptions of 'the machine's context'; at the same time, an understanding of 'the context' derives from a sense of the machine in its context. The sense of context and machine mutually elaborate each other.

For that aspect of context called the user, the reflexive tie is especially marked. The capacity and boundedness of the machine take their sense and meaning from the capacity and boundedness of the user." (Woolgar 1991)

Proposed Amendments to the classical view



A revised picture



Conclusion and open questions

Programming involves two fundamental acts prior to « design » and « implementation »:

1. **Defining the scope of your problem – or its context**
   - It is well known that any plan takes meaning only in a given context, but programming is a kind of planification which aims to explicit its context as fully as possible
   - It includes all resources and goals – humans and machines
2. **Defining the limit between the formal and the informal inside the context – which is the specification**
   - You need to « program » both realms, but in different guises:
   - The formal realm is what the programmer takes full responsiblity of, where he is ready to offer a guarantee of realization of the specification
   - The informal realm is what the programmer assumes the behavior of, where he can only offer descriptions of expected behaviors (whether human, machine or thing)

Design and implementation have two sides because of these two realms

These points seem to suggest a broader epistemic capability, close to « problem-solving » or « instrumental rationality » as a basic human attitude to the world