The Rôle of Domain Engineering in Software Development
and: Why Current Requirements Engineering is Fundamentally Wrong!

PSI’09: 15–19 June 2009

Dines Bjørner
Fredsvej 11, DK-2840 Holte, Danmark
E–Mail: bJORner@gmail.com, URL: www.imm.dtu.dk/~db

April 2, 2009: 12:06
Summary

General

• We introduce the notion of domain descriptions (D) in order to ensure
  ★ that software (S) is right and
  ★ is the right software,
  ★ that is, that it is correct with respect to written requirements (R)
  ★ and that it meets customer expectations (D).
That is, before software can be designed (S)
we must make sure we understand the requirements (R),
and before we can express the requirements
we must make sure that we understand the application domain (D):
- the area of activity of the users of the required software,
- before and after installment of such software.
Informal and Formal Descriptions

• We shall outline what we mean by
  ★ informal, narrative
  ★ and formal domain description,

• and how one can systematically,
  ★ albeit not (in fact: never) automatically
  ★ go from domain descriptions to requirements prescriptions.

• As it seems that domain engineering is a relatively new discipline
  ★ within software engineering
  ★ we shall mostly focus on domain engineering and discuss its necessity.
Summary

Professional Software Engineering

• The talk will show some formulas
  ★ but they are really not meant to be read by the speaker,
  ★ let alone understood, during the talk, by the listeners.
• They are merely there to bring home the point:
  ★ Professional software engineering,
  ★ like other professional engineering branches
  ★ rely on and use mathematics.
• And it is all very simple to learn and practise anyway!
[Summary]

Why Current Requirements Engineering is Flawed!

• We end this talk with, to some, perhaps, controversial remarks:
  ⋆ Requirements engineering, as pursued today,
    ◊ researched, taught and practised,
  ⋆ is outdated, is thus fundamentally flawed.

• We shall justify this claim.
The Software Development Dogma

The Dogma

• The dogma is this:
  ★ Before software can be designed
  ★ we must understand the requirements.
  ★ Before requirements can be finalised
  ★ we must have understood the domain.

• We assume that the audience knows what is meant by
  ★ software design and ★ requirements.

• But what do we mean by “the domain”?
What Do We Mean by ‘Domain’?

• By a domain we shall loosely understand an ‘area’ of
  * natural or
  * human

activity, or both,
where the ‘area’ is “well-delineated” such as, for example,

- for physics:
  - mechanics or chemistry or
  - electricity or hydrodynamics;

- or for an infrastructure component:
  - banking,
  - railways,
  - hospital health-care,
  - “the market”: consumers, retailers, wholesalers,
  - producers and the distribution chain.
By a *domain* we shall thus, less loosely, understand

- a universe of discourse, small or large, a structure
  - (i) of *entities*, that is, of “things”, individuals, particulars
    - some of which are designated as state components;
  - (ii) of *functions*, say over entities,
    - which when applied become possibly state-changing actions of
      the domain;
  - (iii) of *events*,
    - possibly involving entities, occurring in time and
      expressible as predicates over single or pairs of (before/after)
      states; and
  - (iv) of *behaviours*,
    - sets of possibly interrelated sequences of actions and events.
The Role of Domain Engineering in Software Development

[ The Software Development Dogma ]

**Dialectics**

- Now, let’s get this “perfectly” straight!
  - Can we develop software requirements without understanding the domain?
  - Well, how much of the domain should we understand?
  - And how well should we understand it?
• Can we develop software requirements without understanding the domain?

★ No, of course we cannot!
★ But we, you, do develop software for hospitals (railways, banks) without understanding health-care (transportation, the financial markets) anyway!
★ In other engineering disciplines professionalism is ingrained:
  ◇ Aeronautics engineers understand the domain of aerodynamics;
  ◇ naval architects (i.e., ship designers) understand the domain of hydrodynamics;
  ◇ telecommunications engineers understand the domain of electromagnetic field theory;
  ◇ and so forth.
• Well, how much of the domain should we understand?
  ★ A basic answer is this:
  ◊ enough for us to understand formal descriptions of such a domain.
  ★ This is so in classical engineering:
  ◊ Although the telecommunications engineer has not herself researched and made mathematical models of electromagnetic wave propagation in the form of Maxwell’s equations:
    ◊ Gauss’s Law for Electricity,
    ◊ Gauss’s Law for Magnetism,
    ◊ Faraday’s Law of Induction,
    ◊ Ampères Law:
    \[
    \int E \cdot d\vec{A} = \frac{q}{\varepsilon_0} \quad \int B \cdot d\vec{A} = 0 \quad \int E \cdot d\vec{s} = -\frac{d\Phi}{dt} \quad \int B \cdot d\vec{s} = \mu_0 i + \frac{1}{c^2} \frac{\partial}{\partial t} \int E \cdot d\vec{A}
    \]
    ◊ the telecommunications engineer certainly understands these laws.
And how well should we understand it?

★ Well, enough, as an engineer, to manipulate the formulas,
★ to further develop these for engineering calculations.

Conclusion

It is about time that software engineers
★ consult precise descriptions,
★ including formalisations
★ of the application domains for software.

These domain models may have to be developed by computing scientists.

Software engineers then “transform” these into
★ requirements prescriptions
★ and software designs.
The Triptych of Software Development

• We recall the dogma:

★ before software can be designed
★ we must understand the requirements.
★ Before requirements can be finalised
★ we must have understood the domain.

• We conclude from that, that an “ideal” software development proceeds, in three major development phases, as follows:

★ Domain engineering
★ Requirements engineering
★ Software design
### Domain Engineering
- Stakeholder Identification
- Domain Acquisition & Analysis
- Rough Sketching & Terminology
- First Rough Validation

### Requirements Engineering
- Stakeholder Identification
- Domain Acquisition & Analysis
- Rough Sketching & Terminology
- First Rough Validation

### Domain Modelling
- Business Processes
- Intrinsics
- Support Technologies
- Mgt. & Org.
- Rules & Regulations
- Scripts, Lics., Contracts
- Human Behaviour
- Consolidation

### Domain Requirements
- Projection
- Instantiation
- Determination
- Extension
- Fitting
- Consolidation

### Interface Requirements
- Shared Entities
- Shared Operations
- Shared Events
- Shared Behaviours

### Machine Requirements
- Performance
- Dependabilities
- Platform
- Maintenance
- Documentation

### Domain Verification

### Domain Validation

### Domain Theory Formation

### Software Design
- IT System Design
- Software Architecture Design
- Component Design
- Module Design
- Code Design

---

© Dines Björner 2008, Fredsvej 11, DK–2840 Holte, Denmark

April 2, 2009, 12:06
The Triptych of Software Development

The Phase Results

• **Domain engineering**: The results of domain engineering include a domain model: a description,
  ⋆ both informal, as a precise narrative,
  ⋆ and formal, as a specification.

• **Requirements engineering**: The results of requirements engineering include a requirements model: a prescription,
  ⋆ both informal, as a precise narrative,
  ⋆ and formal, as a specification.

• **Software design**: The results of software design include
  ⋆ executable code
  ⋆ and all documentation that goes with it.
[ The Triptych of Software Development ]

Relations to “Reality” and Phase Interrelations

- **Domain engineering:**
  - ★ The domain is described **as it is.**

- **Requirements engineering**
  - ★ The requirements are described **as we would like the software to be,**
  - ★ and the requirements must be clearly related to the domain description.

- **Software design**
  - ★ The software design specification must be **correct with respect to the requirements.**
Below we outline techniques of domain engineering. But just as a preview:

- Based on extensive domain acquisition and analysis
- an informal and a formal domain model is established, a model which is centered around sub-models of:
  - intrinsics,
  - supporting technologies,
  - mgt. and org.,
  - rules and regulations,

which are then

- validated and verified.

- script [or contract] languages and
- human behaviours,
Requirements Engineering

• Below we outline techniques of requirements engineering. But just as a preview:

  ★ Based on presentations of the domain model to requirements stakeholders
  ★ requirements can now be “derived” from the domain model and as follows:

  ◊ First a domain requirements model:
     ○ projection,
     ○ instantiation,
     ○ determination,
     ○ extension and
  ○ fitting of several, separate domain requirements models;
  ◊ then an interface requirements model,
  ◊ and finally a machine requirements model.

  ★ These are simultaneously verified and validated
  ★ and the feasibility and satisfiability of the emerging model is checked.
Software Design

- We do not cover techniques of software design in detail — so only this summary.

  ★ From the requirements prescription one develops,
  ◦ in stages and steps of transformation ("refinement"),
  ◦ first the system architecture,
  ◦ then the program (code) organisation (structure),
  ◦ and then, in further steps of development,
    ◦ the component design,
    ◦ the module design and
    ◦ the code.

  ★ These stages and step can be verified, model checked and tested with respect
    ◦ to the previous phase of requirements prescription,
    ◦ respectively the previous software design stages and steps.

- One can then assert that the Software design is correct with respect to the
  Requirements in the context of the assumptions expressed about the Domain:

  \[ \mathcal{D}, \mathcal{S} \models \mathcal{R} \]
PSI’09: 15–19 June 2009 and: Why Current Requirements Engineering is Fundamentally Wrong!

### Domain Engineering
- Stakeholder Identification
- Domain Acquisition & Analysis
- Rough Sketching & Terminology
- First Rough Validation

### Domain Modelling
- Business Processes
- Intrinsics
- Support Technologies
- Mgt. & Org.
- Rules & Regulations
- Scripts, Lics., Contracts
- Human Behaviour
- Consolidation

### Domain Verification
- Domain Validation
- Domain Theory Formation

### Requirements Engineering
- Stakeholder Identification
- Domain Acquisition & Analysis
- Rough Sketching & Terminology
- First Rough Validation

### Domain Requirements
- Projection
- Instantiation
- Determination
- Extension
- Fitting
- Consolidation

### Domain Validation
- Domain Verification

### Domain Satisfiability
- Requirements Verification
- Requirements Validation
- Requirements Feasibility
- Requirements Satisfiability

### Domain Verification
- Domain Verification
- Domain Theory Formation

### Software Design
- IT System Design
- Software Architecture Design
- Component Design
- Module Design
- Code Design

© Dines Bjørner 2008, Fredsvej 11, DK–2840 Holte, Denmark
Domain Engineering

• We shall focus only on the actual modelling, thus omitting any treatment of
  ★ the preparatory administrative and informative work,
  ★ the identification of and liaison with domain stakeholders,
  ★ the domain acquisition and analysis, and
  ★ the establishment of a domain terminology (document).

• So we go straight to the descriptive work.
  ★ We first illustrate the ideas of modelling domain phenomena and concepts in terms of simple entities, operations, events and behaviours,
  ★ then we model the domain in terms of domain facets.

• Also, at then end, we do not have time and paper space for any treatment of domain verification, domain validations and the establishment of a domain theory.
[ Domain Engineering ]

Simple Entities, Operations, Events and Behaviours

- Without discussing our specification ontology,
  - that is, the principles according to which we view the world around us,

- we just present the decomposition of phenomena and concepts into
  - simple entities,
  - operations,
  - events and
  - behaviours.

- All of these are “first class citizens”, that is, are entities.

- We now illustrate examples of each of these ontological categories.
Simple Entities

• A *simple entity* is something that has a distinct, separate existence, though it need not be a material existence, to which we apply functions.

• With simple entities we associate attributes, i.e., properties modelled as types and values.

• Simple entities can be considered
  ★ either continuous
  ★ or discrete,
    ◊ and, if discrete
      ○ then either atomic
      ○ or composite.
It is the observer (that is, the specifier) who decides whether to consider a simple entity to be atomic or composite.

Atomic entities cannot meaningfully be decomposed into sub-entities, but atomic entities may be analysed into (Cartesian) “compounds” of properties, that is, attributes. Attributes have name, type and value.

Composite entities can be meaningfully decomposed into sub-entities, which are entities.

The composition of sub-entities into a composite entity “reveals” the, or a mereology of the composite entity: that is, how it is “put together”.

Example 1: Transport Entities: Nets, Links and Hubs — Narrative

1. There are hubs and links.

2. There are nets, and a net consists of a set of two or more hubs and one or more links.

3. There are hub and link identifiers.

4. Each hub (and each link) has an own, unique hub (respectively link) identifiers (which can be observed from the hub [respectively link]).
Example 2: Transport Entities: Nets, Links and Hubs — Formalisation

type
1 H, L,
2 N = H-set \times L-set

axiom
2 \forall (hs,ls):N \cdot \text{card } hs \geq 2 \land \text{card } ks \geq 1

type
3 HI, LI

value
4a \text{obs}_HI: H \rightarrow HI, \text{obs}_LI: L \rightarrow LI

axiom
4b \forall h,h':H, l,l':L \cdot h \neq h' \Rightarrow \text{obs}_HI(h) \neq \text{obs}_HI(h') \land l \neq l' \Rightarrow \text{obs}_LI(l) \neq \text{obs}_LI(l')
Operations

- By an operation we shall understand something which when applied to some entities, called the arguments of the operation, yields an entity, called the result of the operation application (also referred to as the operation invocation).

  ★ Operations have signatures, that is, can be grossly described by the Cartesian type of its arguments and the possibly likewise compounded type of its results.

  ★ Operations may be total over their argument types, or may be just partial. We shall consider some acceptable operations as “never terminating” processes.

  ★ We shall, for the sake of consistency, consider all operation invocations as processes (terminating or non-terminating), and shall hence consider all operation definitions as also designating process definitions.
We shall also use the term **function** to mean the same as the term operation.

By a **state** we shall loosely understand a collection of one or more simple entities whose value may change.

By an **action** we shall understand an operation application which applies to and/or yields a state.
Example 3: Link Insertion Operation

5. To a net one can insert a new link in either of three ways:

(a) Either the link is connected to two existing hubs — and the insert operation must therefore specify the new link and the identifiers of two existing hubs;

(b) or the link is connected to one existing hub and to a new hub — and the insert operation must therefore specify the new link, the identifier of an existing hub, and a new hub;

(c) or the link is connected to two new hubs — and the insert operation must therefore specify the new link and two new hubs.

(d) From the inserted link one must be able to observe identifier of respective hubs.

6. From a net one can remove a link. The removal command specifies
a link identifier.

**type**

5 Insert == Ins(s_ins:Ins)  
5 Ins = 2xHubs | 1x1nH | 2nHs  
5(a) 2xHubs == 2oldH(s_hi1:HI,s_l:L,s_hi2:HI)  
5(b) 1x1nH == 1oldH1newH(s_hi:HI,s_l:L,s_h:H)  
5(c) 2nHs == 2newH(s_h1:H,s_l:L,s_h2:H)

**axiom**

5(d) \( \forall \exists 2\text{oldH}(hi',l,hi''):\text{Ins} \cdot hi' \neq hi'' \wedge \text{obs}_LIs(l)=\{hi',hi''\} \wedge \forall 1\text{old1newH}(hi,l,h):\text{Ins} \cdot \text{obs}_LIs(l)=\{hi,\text{obs}_HI(h)\} \wedge \forall 2\text{newH}(h',l,h''):\text{Ins} \cdot \text{obs}_LIs(l)=\{\text{obs}_HI(h'),\text{obs}_HI(h'')\} \)

7. If the **Insert** command is of kind **2newH**(h',l,h'') then the updated net of hubs and links, has

  - the hubs hs joined, \( \cup \), by the set \{h',h''\} and
8. If the **insert** command is of kind `1oldH1newH(hi,l,h)` then the updated net of hubs and links, has

8.1 : the hub identified by `hi` updated, `hi'`, to reflect the link connected to that hub.

8.2 : The set of hubs has the hub identified by `hi` replaced by the updated hub `hi'` and the new hub.

8.2 : The set of links augmented by the new link.

9. If the **insert** command is of kind `2oldH(hi',l,hi'')` then

9.1–.2 : the two connecting hubs are updated to reflect the new link,

9.3 : and the resulting sets of hubs and links updated.

\[
\text{int\_insert}(op)(hs,ls) \equiv \\
\star_i \text{ case } op \text{ of } \\
7 \quad 2\text{newH}(h',l,h'') \rightarrow (hs \cup \{h',h''\},ls \cup \{l\}),
\]
8 \text{oldH1newH}(hi,l,h) \rightarrow
8.1 \textbf{let} h' = aLI(xtr_H(hi,hs),obs_{LI}(l)) \textbf{ in}
8.2 (hs \setminus \{xtr_H(hi,hs)\} \cup \{h,h'\},ls \cup \{l\}) \textbf{ end,}
9 \text{oldH}(hi',l,hi'') \rightarrow
9.1 \textbf{let} hs\delta = \{aLI(xtr_H(hi',hs),obs_{LI}(l)),
9.2 aLI(xtr_H(hi'',hs),obs_{LI}(l))\} \textbf{ in}
9.3 (hs \setminus \{xtr_H(hi',hs),xtr_H(hi'',hs)\} \cup hs\delta,ls \cup \{l\}) \textbf{ end}
\star_j \textbf{ end} \quad \star_k \textbf{ pre}_{pre\_int\_Insert(op)}(hs,ls)
Events

• Informally, by an event we shall loosely understand the occurrence of “something” that may either trigger an action, or is triggered by an action, or alter the course of a behaviour, or a combination of these.

• An event can be characterised by
  ⋆ a predicate, \( p \) and
  ⋆ a pair of (“before”) and (“after”) of pairs of
    ◇ states and
    ◇ times:
      ◇ \( p( (t_b, \sigma_b), (t_a, \sigma_a)) \).

  ⋆ Usually the time interval \( t_a - t_b \)
  ⋆ is of the order \( t_a \simeq (t_b) + \delta_{\text{tiny}} \).
Example 4: Transport Events

• (i) A link, for some reason “ceases to exist”; for example:
  ★ a bridge link falls down,
  ★ or a level road link is covered by a mud slide,
  ★ or a road tunnel is afire,
  ★ or a link is blocked by some vehicle accident.

• (ii) A vehicle enters or leaves the net.

• (iii) A hub is saturated with vehicles.
[ Domain Engineering, Simple Entities, Operations, Events and Behaviours ]

Behaviours

• By a behaviour we shall informally understand a strand of (sets of) actions and events.

  ★ In the context of domain descriptions we shall speak of behaviours
  ★ whereas, in the context of requirements prescriptions and software designs we
    shall use the term processes.

• By a behaviour we, more formally, understand a sequence, $q$

  ★ of actions ★ and/or events

  $q_1, q_2, \ldots, q_i, q_{i+1}, \ldots, q_n$

• such that the state

  ★ resulting from one such action, $q_i$, ★ or in which some event, $q_i$, occurs,

• becomes the state in which the next action or event, $q_{i+1}$,

  ★ if it is an action, is effected, ★ or, if it is an event, is the event state.
Example 5: Transport: Traffic Behaviour

10. There are further undefined vehicles.
11. Traffic is a discrete function from a ‘Proper subset of Time’ to pairs of nets and vehicle positions.
12. Vehicles positions is a discrete function from vehicles to vehicle positions.

\[
\begin{align*}
10 & \quad \text{Veh} \\
11 & \quad \text{TF} = \text{Time} \xrightarrow{m} (\mathbb{N} \times \text{VehPos}) \\
12 & \quad \text{VehPos} = \text{Veh} \xrightarrow{m} \text{Pos}
\end{align*}
\]
There are positions, and a position is either on a link or in a hub.

(a) A hub position is indicated just by a triple: the identifier of the hub in question, and a pair of (from and to) link identifiers, namely of links connected to the identified hub.

(b) A link position is identified by a quadruplet: The identifier of the link, a pair of hub identifiers (of the link connected hubs), designating a direction, and a real number, properly between 0 and 1, denoting the relative offset from the from hub to the to hub.

\[
\text{type} \\
13 \ Pos = HPos \ | \ LPos \\
13(a) \ HPos == hpos(s_{hi}:HI,s_{fli}:LI,s_{tli}:LI) \\
13(b) \ LPos == lpos(s_{li}:HI,s_{fhi}:LI,s_{tli}:LI,s_{offset}:Frac) \\
13(b) \ Frac = \{ |r:\text{Real}\cdot0<r<1| \}
\]
### Domain Engineering
- Stakeholder Identification
- Domain Acquisition & Analysis
- Rough Sketching & Terminology
- First Rough Validation
- Domain Theory Formation

### Requirements Engineering
- Stakeholder Identification
- Domain Acquisition & Analysis
- Rough Sketching & Terminology
- First Rough Validation

### Domain Modelling
- Business Processes
  - Intrinsics
  - Support Technologies
  - Mgt. & Org.
  - Rules & Regulations
  - Scripts, Lics., Contracts
  - Human Behaviour
- Domain Verification
- Domain Validation

### Domain Requirements
- Projection
- Instantiation
- Determination
- Extension
- Fitting
- Consolidation

### Interface Requirements
- Shared Entities
- Shared Operations
- Shared Events
- Shared Behaviours

### Machine Requirements
- Performance
- Dependabilities
- Platform
- Maintenance
- Documentation

### Software Design
- IT System Design
- Software Architecture Design
- Component Design
- Module Design
- Code Design

### Consolidation
- Scripts, Lics., Contracts
- Rules & Regulations
- Mgt. & Org.
- Human Behaviour
- Domain Modelling

### Domain Verification
- Requirements Verification
- Requirements Validation

### Domain Validation
- Requirements Feasibility
- Requirements Satisfiability
By a **domain facet** we mean

- one amongst a finite set of generic ways
- of analysing a domain:
- a view of the domain,
- such that the different facets cover conceptually different views,
- and such that these views together cover the domain

We shall postulate the following domain facets:

- intrinsics,
- support technologies,
- management & organisation,
- rules & regulations,
- script languages [contract languages] and
- human behaviour.

Each facet covers simple entities, operations, events and behaviours.

We shall now illustrate these.
PSI’09: 15–19 June 2009 and: Why Current Requirements Engineering is Fundamentally Wrong!

Domain Engineering
- Stakeholder Identification
- Domain Acquisition & Analysis
- Rough Sketching & Terminology
- First Rough Validation

Domain Modelling
- Business Processes
  - Intrinsic
- Support Technologies
- Mgt. & Org.
- Rules & Regulations
- Scripts, Lics., Cntrs.
- Human Behaviour
- Consolidation
- Domain Verification
- Domain Validation
- Domain Theory Formation

Requirements Engineering
- Stakeholder Identification
- Domain Acquisition & Analysis
- Rough Sketching & Terminology
- First Rough Validation

Domain Requirements
- Projection
- Instantiation
- Determination
- Extension
- Fitting
- Consolidation

Interface Requirements
- Shared Entities
- Shared Operations
- Shared Events
- Shared Behaviours

Machine Requirements
- Performance
- Dependabilities
- Platform
- Maintenance
- Documentation

- Requirements Verification
- Requirements Validation
- Requirements Feasibility
- Requirements Satisfiability
Intrinsics

By **domain intrinsics** we mean

\* those phenomena and concepts of a domain which are basic to any of the other facets (listed earlier and treated, in some detail, below),

\* with such domain intrinsics initially covering at least one specific, hence named, stakeholder view.

---

**Example 6: Intrinsics, I**

- The links, hubs, hence the nets,
- and the identifiers of links and hubs
- are intrinsic phenomena, respectively concepts.

- So are:
Example 7: Intrinsic, II

14. From any link of a net one can observe the two hubs to which the link is connected.
   (a) We take this ‘observing’ to mean the following: From any link of a net one can observe the two distinct identifiers of these hubs.

15. From any hub of a net one can observe the one or more links to which are connected to the hub.
   (a) Again: by observing their distinct link identifiers.

16. Extending Item 14: the observed hub identifiers must be identifiers of hubs of the net to which the link belongs.

17. Extending Item 15: the observed link identifiers must be identifiers of links of the net to which the hub belongs.
value
14a  \( \text{obs}_{\text{HIs}} : L \rightarrow \text{HI-set} \),
15a  \( \text{obs}_{\text{LIs}} : H \rightarrow \text{LI-set} \),

axiom
14b  \( \forall l:L \cdot \text{card} \ \text{obs}_{\text{HIs}}(l) = 2 \land 
\)
15b  \( \forall h:H \cdot \text{card} \ \text{obs}_{\text{LIs}}(h) = 1 \land 
\)
\( \forall (hs,ls):N \cdot 
\)
14(a)  \( \forall h:H \cdot h \in hs \Rightarrow \forall li:LI \cdot li \in \text{obs}_{\text{LIs}}(h) \Rightarrow 
\)
\( \exists l':L \cdot l' \in ls \land li=\text{obs}_{\text{LIs}}(l') \land \text{obs}_{\text{HI}}(h) \in \text{obs}_{\text{HIs}}(l') \land 
\)
15(a)  \( \forall l:L \cdot l \in ls \Rightarrow 
\)
\( \exists h',h'':H \cdot \{h',h''\} \subseteq hs \land \text{obs}_{\text{HIs}}(l) = \{\text{obs}_{\text{HI}}(h'),\text{obs}_{\text{HI}}(h'')\} \)
16  \( \forall h:H \cdot h \in hs \Rightarrow \text{obs}_{\text{LIs}}(h) \subseteq \text{iols}(ls) \)
17  \( \forall l:L \cdot l \in ls \Rightarrow \text{obs}_{\text{HIs}}(h) \subseteq \text{iohs}(hs) \)

value
\( \text{iohs} : \text{H-set} \rightarrow \text{HI-set} \), \( \text{iols} : \text{L-set} \rightarrow \text{LI-set} \)
\( \text{iohs}(hs) \equiv \{\text{obs}_{\text{HI}}(h) | h:H \cdot h \in hs\} \)
\( \text{iols}(ls) \equiv \{\text{obs}_{\text{LI}}(l) | l:L \cdot l \in ls\} \)
Stakeholder Identification

Domain Acquisition & Analysis

Rough Sketching & Terminology

First Rough Validation

Domain Theory Formation

Stakeholder Identification

Domain Acquisition & Analysis

Rough Sketching & Terminology

First Rough Validation

Domain Requirements

Projection
Instantiation
Determination
Extension
Fitting
Consolidation

Interface Requirements

Shared Entities
Shared Operations
Shared Events
Shared Behaviours

Machine Requirements

Performance
Dependabilities
Platform
Maintenance
Documentation

Requirements Verification

Requirements Validation

Requirements Feasibility

Requirements Satisfiability

Support Technologies

Business Processes
Intrinsics

Mgt. & Org.
Rules & Regulations
Scripts, Lics., Cntrs.
Human Behaviour
Consolidation

Consolidation

Domain Modelling

Domain Requirements

Domain Engineering

Requirements Engineering

Domain Verification

Domain Validation
By **domain support technologies** we mean

- ways and means of concretesing
- certain observed (abstract or concrete) phenomena or
- certain conceived concepts
- in terms of (possibly combinations of)

- human work,
- mechanical,
- hydro mechanical,
- thermo-mechanical,
- pneumatic,
- aero-mechanical,
- electro-mechanical,
- electrical,
- electronic,
- telecommunication,
- photo/opto-electric,
- chemical, etc.

(possibly computerised) sensor, actuator tools.
In this example of a support technology

* we shall illustrate an abstraction
* of the kind of semaphore signalling
* one encounters at road intersections, that is, hubs.

The example is indeed an abstraction:

* we do not model the actual “machinery”
  ♦ of road sensors,
  ♦ hub-side monitoring & control boxes, and
  ♦ the actuators of the green/yellow/red semaphore lamps.
* But, eventually, one has to,
* all of it,
* as part of domain modelling.
Example 8: Hub Sempahores

- To model signalling we need to model hub and link states.

- A hub (link) state is the set of all traversals that the hub (link) allows.
  
  - A hub traversal is a triple of identifiers:
      - of the link from where the hub traversal starts,
      - of the hub being traversed, and
      - of the link to where the hub traversal ends.
  
  - A link traversal is a triple of identifiers:
      - of the hub from where the link traversal starts,
      - of the link being traversed, and
      - of the hub to where the link traversal ends.
  
  - A hub (link) state space is the set of all states that the hub (link) may be in.
  
  - A hub (link) state changing operation can be designated by
      - the hub and a possibly new hub state (the link and a possibly new link state).

\[
\begin{align*}
\text{type} \quad & L\Sigma' = L_{\text{Trav}-\text{set}} \\
L_{\text{Trav}} & = (HI \times LI \times HI) \\
L\Sigma & = \{| \lnk\sigma : L\Sigma' \cdot \text{syn}_{\text{wf}} L\Sigma | \lnk\sigma \} |
\end{align*}
\]
HΣ' = H\_Trav\_set
H\_Trav = (L\_1 \times H\_1 \times L\_1)
HΣ = \{\mid \text{hub}\sigma:HΣ' \cdot \text{wf}_{HΣ}\{\text{hub}\sigma}\} \}
HΩ = HΣ\_set, LΩ = LΣ\_set

value
obs\_LΣ: \text{L} \rightarrow LΣ, obs\_LΩ: \text{L} \rightarrow LΩ
obs\_HΣ: \text{H} \rightarrow HΣ, obs\_HΩ: \text{H} \rightarrow HΩ

axiom
\forall h: \text{H} \cdot \text{obs\_HΣ}(h) \in \text{obs\_HΩ}(h) \land \forall l: \text{L} \cdot \text{obs\_LΣ}(l) \in \text{obs\_LΩ}(l)

value
\text{chg\_HΣ}: \text{H} \times HΣ \rightarrow \text{H}, \text{chg\_LΣ}: \text{L} \times LΣ \rightarrow \text{L}
\text{chg\_HΣ}(h,h\sigma) \text{ as } h' \\
\text{pre } h\sigma \in \text{obs\_HΩ}(h) \text{ post } \text{obs\_HΣ}(h') = h\sigma
\text{chg\_LΣ}(l,l\sigma) \text{ as } l' \\
\text{pre } l\sigma \in \text{obs\_LΩ}(h) \text{ post } \text{obs\_HΣ}(l') = l\sigma

- Well, so far we have indicated that there is an operation that can change hub and link states.
- But one may debate whether those operations shown are really examples of a support technology.
  (That is, one could equally well claim that they remain examples of intrinsic facets.)
- We may accept that and then ask the question:
★ How to effect the described state changing functions?
★ In a simple street crossing a semaphore does not instantaneously change from red to green in one direction while changing from green to red in the cross direction.
★ Rather there is are intermediate sequences of, for example, not necessarily synchronised green/yellow/red and red/yellow/green states to help avoid vehicle crashes and to prepare vehicle drivers.

- Our “solution” is to modify the hub state notion.

**type**

<table>
<thead>
<tr>
<th>Colour == red</th>
<th>yellow</th>
<th>green</th>
</tr>
</thead>
</table>

\[
X = LI \times HI \times LI \times Colour \ [crossings \ \text{of} \ \text{a hub}]
\]

\[
H\Sigma = X\text{-set} \ [\text{hub states}]
\]

**value**

\[
\text{obs}_H\Sigma: H \rightarrow H\Sigma, \ xtr_\text{Xs}: H \rightarrow X\text{-set}
\]

\[
\text{xtr}_\text{Xs}(h) \equiv
\]

\[
\{(li,hi,li',c)|li,li':LI,hi:HI,c:Colour\cdot\{li,li'\} \subseteq \text{obs}_L\text{Is}(h)\land hi=\text{obs}_HI(h)\}
\]

**axiom**

\[
\forall \ n:N,h:H \cdot h \in \text{obs}_H\text{s}(n) \Rightarrow \text{obs}_H\Sigma(h) \subseteq \text{xtr}_\text{Xs}(h) \land
\]

\[
\forall \ (li1,hi2,li3,c),(li4,hi5,li6,c'):\ X \cdot
\]

\[
\{(li1,hi2,li3,c),(li4,hi5,li6,c')\} \subseteq \text{obs}_H\Sigma(h) \land
\]
\[ \text{li1}=\text{li4} \wedge \text{hi2}=\text{hi5} \wedge \text{li3}=\text{li6} \Rightarrow \text{c}=\text{c}' \]

- We consider the colouring, or any such scheme, an aspect of a support technology facet.
- There remains, however, a description of how the technology that supports the intermediate sequences of colour changing hub states.
- We can think of each hub being provided with a mapping from pairs of “stable” (that is non-yellow coloured) hub states \((h\sigma_i, h\sigma_f)\) to well-ordered sequences of intermediate “un-stable” (that is yellow coloured) hub states
  - \(\ast\) paired with some time interval information
  - \(\ast\) \(\langle (h\sigma', t\delta'), (h\sigma'', t\delta''), \ldots, (h\sigma''', t\delta'''') \rangle\)
  - \(\ast\) and so that each of these intermediate states can be set,
  - \(\ast\) according to the time interval information,\(^1\)
  - \(\ast\) before the final hub state \((h\sigma_f)\) is set.

\textbf{type}

\begin{align*}
\text{TI [time interval]} \quad \\
\text{Signalling} = (H\Sigma \times \text{TI})^* \\
\text{Sema} = (H\Sigma \times H\Sigma) \xrightarrow{m} \text{Signalling}
\end{align*}

\textbf{value}

\begin{align*}
\text{obs\_Sema}: H \rightarrow \text{Sema}, \quad \text{chg\_H\Sigma}: H \times H\Sigma \rightarrow H, \quad \text{chg\_H\Sigma\_Seq}: H \times H\Sigma \rightarrow H
\end{align*}
chg\_HΣ(h,hσ) as h’ pre hσ ∈ obs\_HΩ(h) post obs\_HΣ(h’) = hσ

\begin{align*}
\text{let } \text{sigseq} &= (\text{obs\_Sema}(h))(\text{obs\_Σ}(h),hσ) \text{ in sig\_seq}(h)(\text{sigseq}) \text{ end} \\
\text{sig\_seq}: H \rightarrow \text{Signalling} \rightarrow H \\
\text{sig\_seq}(h)(\text{sigseq}) &\equiv \\
\text{if } \text{sigseq}=\langle \rangle \text{ then } h \text{ else} \\
\text{let } (hσ,tδ) &= \text{hd sigseq in} \\
\text{let } h’ &= \text{chg\_HΣ}(h,hσ); \text{ wait tδ;} \\
\text{sig\_seq}(h’)(\text{tl sigseq}) \text{ end end end}
\end{align*}
Stakeholder Identification

Domain Acquisition & Analysis

Rough Sketching & Terminology

First Rough Validation

Domain Theory Formation

Domain Engineering

Requirements Engineering

Domain Identification

Requirements Acquisition & Analysis

Requirements Rough Sketch & Terminology

Requirements First Rough Validation

Domain Requirements

Domain Modelling

Business Processes

Intrinsics

Support Technologies

Mgt. & Org.

Rules & Regulations

Scripts, Lics., Contracts

Human Behaviour

Consolidation

Domain Verification

Domain Validation

Requirements Feasibility

Requirements Satisfaction

Projection

Instantiation

Determination

Extension

Fitting

Consolidation

Interface Requirements

Shared Entities

Shared Operations

Shared Events

Shared Behaviours

Machine Requirements

Performance

Dependabilities

Platform

Maintenance

Documentation

Requirements Verification

Requirements Validation

Requirements Feasibility

Requirements Satisfiability

Projection

Instantiation

Determination

Extension

Fitting

Consolidation

Shared Entities

Shared Operations

Shared Events

Shared Behaviours

Performance

Dependabilities

Platform

Maintenance

Documentation

Requirements Verification

Requirements Validation

Requirements Feasibility

Requirements Satisfiability
By domain management we mean people

- (i) who determine, formulate and thus set standards (cf. rules and regulations, a later lecture topic) concerning strategic, tactical and operational decisions;
- (ii) who ensure that these decisions are passed on to (lower) levels of management, and to “floor” staff;
- (iii) who make sure that such orders, as they were, are indeed carried out;
- (iv) who handle undesirable deviations in the carrying out of these orders cum decisions;
- and (v) who “backstop” complaints from lower management levels and from floor staff.
Organisation

• By domain organisation we mean
  ★ the structuring of management and non-management staff levels;
  ★ the allocation of
    ◊ strategic, tactical and operational concerns
    ◊ to within management and non-management staff levels;
  ★ and hence the “lines of command”:
    ◊ who does what and
    ◊ who reports to whom —
      ◦ administratively and
      ◦ functionally.
Examples

Example 9: Bus Transport Management & Organisation

- On Slides 78–84 we illustrate what is there called a contract language.
  - “Programs” in that language are either contracts or are orders to perform the actions permitted or obligated by contracts.
  - The language in question is one of managing bus traffic on a net.
  - The management & organisation of bus traffic involves
    - contractors issuing contracts,
    - contractees acting according to contracts,
    - busses (owned or leased) by contractees,
    - and the bus traffic on the (road) net.
  - Contractees, i.e., bus operators,
"start" buses according to a contract timetable,
"cancel" buses if and when deemed necessary,
"insert" rush-hour and other buses if and when deemed necessary,
and, acting as contractors, "sub-contract" sub-contractees to operate bus lines,
for example, when the issuing contractor is not able to operate these bus lines,
i.e., not able to fulfill contractual obligations,
due to unavailability of buses or staff.

Clearly the programs of bus contract languages
★ are “executed” according to management decisions
★ and the sub-contracting “hierarchy” reflects organisational facets.
Human stakeholders act in the domain, whether

★ clients,
★ workers,
★ managers,
★ suppliers,
★ regulatory authorities,
★ or other.

Their actions are guided and constrained by rules and regulations.

These are sometimes implicit, that is, not “written down”.

But we can talk about rules and regulations as if they were explicitly formulated.
The main difference between rules and regulations is that

- rules express properties that must hold and
- regulations express state changes that must be effected if rules are observed broken.

Rules and regulations are directed

- not only at human behaviour
- but also at expected behaviours of support technologies.

Rules and regulations are formulated

- by enterprise staff, management or workers,
- and/or by business and industry associations,
  - for example in the form of binding or guiding
  - national, regional or international standards,
- and/or by public regulatory agencies.
Domain Rules

• By a domain rule we mean
  ★ some text
  ★ which prescribes how people or equipment
  ★ are expected to behave when dispatching their duty,
  ★ respectively when performing their functions.

Domain Regulations

• By a domain regulation we mean
  ★ some text
  ★ which prescribes what remedial actions are to be taken
  ★ when it is decided that a rule has not been followed according to
    its intention.
Two Informal Examples

Example 10: Trains at Stations: Available Station Rule and Regulation

- Rule:
  - In China the arrival and departure of trains at, respectively from, railway stations is subject to the following rule:
  - In any three-minute interval at most one train may either arrive to or depart from a railway station.

- Regulation:
  - If it is discovered that the above rule is not obeyed, then there is some regulation which prescribes administrative or legal management and/or staff action, as well as some correction to the railway traffic.
Example 11: Trains Along Lines: Free Sector Rule and Regulation

- **Rule:**
  - In many countries railway lines (between stations) are segmented into blocks or sectors. The purpose is to stipulate that if two or more trains are moving along the line, then:
    - *There must be at least one free sector (i.e., without a train) between any two trains along a line.*

- **Regulation:**
  - *If it is discovered that the above rule is not obeyed, then there is some regulation which prescribes administrative or legal management and/or staff action, as well as some correction to the railway traffic.*
A Formal Example

• We shall develop the above example (11, Slide 64) into a partial, formal specification.

• That is, not complete, but “complete enough” for the reader to see what goes on.

Example 12: Continuation of Example 11 Slide 64

• We start by analysing the text of the rule and regulation.

  ★ The rule text: *There must be at least one free sector (i.e., without a train) between any two trains along a line.* contains the following terms:

  ◊ free (a predicate), ◊ train (an entity) and
  ◊ sector (an entity), ◊ line (an entity).

• We shall therefore augment our formal model to reflect these terms.

• We start by modelling

  ★ sectors and sector descriptors, ★ trains, and
  ★ lines and train position descriptors, ★ the predicate free.
type

\[ \text{Sect'} = H \times L \times H, \]
\[ \text{SectDescr} = HI \times LI \times HI \]
\[ \text{Sect} = \{(h,l,h'):\text{Sect'} \cdot \text{obs_HI}(l) = \{\text{obs_HI}(h), \text{obs_HI}(h')\}\} \]
\[ \text{SectDescr} = \{(hi,li,hi'):\text{SectDescr'} \cdot \exists (h,l,j'):\text{Sect}\cdot \text{obs_HI}(l) = \{\text{obs_HI}(h), \text{obs_HI}(h')\}\} \]
\[ \text{Line'} = \text{Sect}^*, \]
\[ \text{Line} = \{(line:\text{Line'}\cdot \text{wfLine}(line))\}\]
\[ \text{TrnPos'} = \text{SectDescr}^* \]
\[ \text{TrnPos} = \{(trnpos:\text{TrnPos'})\cdot \exists line:\text{Line}\cdot \text{convLine_to_TrnPos(line)} = trnpos '\}\]

value

\[ \text{wfLine}: \text{Line'} \to \text{Bool} \]
\[ \text{wfLine}(line) \equiv \]
\[ \forall i: \text{Nat} \cdot \{i, i+1\} \subseteq \text{inds}(line) \Rightarrow \]
\[ \text{let } (_, l, h) = \text{line}(i), (h', l', _) = \text{line}(i+1) \text{ in } \text{h} = h' \text{ end} \]
\[ \text{convLine_to_TrnPos}: \text{Line} \to \text{TrnPos} \]
\[ \text{convLine_to_TrnPos}(line) \equiv \]
\[ \langle \langle \text{obs_HI}(h), \text{obs_LI}(l), \text{obs_HI}(h') \rangle | 1 \leq i \leq \text{len line} \land \text{line}(i) = (h, l, h') \rangle \]
value
lines: \( N \rightarrow \text{Line-set} \)
lines(hs,ls) \equiv
\begin{align*}
\text{let } \text{lns} &= \{(h,l,h')|h,h':H,l:L:\text{proper line}((h,l,h'),(hs,ls))\} \\
&\cup \{ln - ln'|ln,l':\text{Line}\cdot\{ln,ln'\} \subseteq \text{lns} \land \text{adjacent}(ln,ln')\} \text{ in} \\
\text{lns} \text{ end}
\end{align*}

adjacent: \text{Line} \times \text{Line} \rightarrow \text{Bool}
adjacent((\_,l,h),(h',\_,\_)) \equiv h = h'
pre \{\text{obs\_LI}(l),\text{obs\_LI}(l')\} \subseteq \text{obs\_LIs}(h)

type
TF = T \xrightarrow{m} (N \times (TN \xrightarrow{m} \text{TrnPos}))

value
wf_TF: TF \rightarrow \text{Bool}
\[
\begin{align*}
\text{wf\_TF}(tf) & \equiv \\
\forall t: T \cdot t \in \text{dom} \; tf \Rightarrow \\
\text{let } ((hs,ls),\text{trnposs}) = tf(t) \text{ in} \\
\forall \text{trn} : TN \cdot \text{trn} \in \text{dom} \; \text{trnposs} \Rightarrow \\
\exists \text{line} : Line \cdot \text{line} \in \text{lines}(hs,ls) \land \\
\text{trnposs}(\text{trn}) = \text{conv\_Line\_to\_TrnPos}(\text{line}) \end{align*}
\]

- Nothing prevents two or more trains from occupying overlapping train positions.
- They have “merely” – and regrettably – crashed. But such is the domain.
- So \(\text{wf\_TF}(tf)\) is not part of an axiom of traffic, merely a desirable property.

**value**

\[
\begin{align*}
\text{has\_free\_Sector} : TN \times T \to TF \to \text{Bool} \\
\text{has\_free\_Sector}(\text{trn},(hs,ls),t)(tf) & \equiv \\
\text{let } ((hs,ls),\text{trnposs}) = tf(t) \text{ in} \\
(\text{trn} \not\in \text{dom} \; \text{trnposs} \lor (\text{tn} \in \text{dom} \; \text{trnposs}(\text{t}) \land \\
\exists \text{ln} : Line \cdot \text{ln} \in \text{lines}(hs,ls) \land \\
\text{is\_prefix}(\text{trnposs}(\text{trn}),\text{ln})(hs,ls)) \land \\
\sim \exists \text{trn}' : TN \cdot \text{trn}' \in \text{dom} \; \text{trnposs} \land \text{trn}' \not= \text{trn} \land \\
\end{align*}
\]
trnposs(trn')=conv_Line_to_TrnPos(⟨follow_Sect(ln)(hs,ls)⟩)
end
pre exists_follow_Sect(ln)(hs,ls)

is_prefix: Line × Line → N → Bool
is_prefix(ln,ln')(hs,ls) ≡ \exists ln'':Line \cdot ln'' ∈ lines(hs,ls) ∧ ln~ln''=ln'

exists_follow_Sect: Line → Net → Bool
exists_follow_Sect(ln)(hs,ls) ≡ \
\exists ln':Line\cdot ln' ∈ lines(hs,ls)∧ln~ln' ∈ lines(hs,ls)
pre ln ∈ lines(hs,ls)

follow_Sect: Line → Net → Sect
follow_Sect(ln)(hs,ls) ≡ 
let ln':Line\cdot ln' ∈ lines(hs,ls)∧ln~ln' ∈ lines(hs,ls) in hd ln' end
pre line ∈ lines(hs,ls)∧exists_follow_Sect(ln)(hs,ls)

• We doubly recursively define a function free_sector_rule(tf)(r).
• \( tf \) is that part of the traffic which has yet to be “searched” for non-free sectors.

★ Thus \( tf \) is “counted” up from a first time \( t \) till the traffic \( tf \) is empty.
★ That is, we assume a finite definition set \( tf \).

• \( r \) is like a traffic but without the net.

★ Initially \( r \) is the empty traffic.
★ \( r \) is “counted” up from “earliest” cases of trains with no free sector ahead of them.

• The recursion stops, for a given time when

★ there are no more train positions to be “searched” for that time;
★ and when the “to-be-searched” traffic is empty.

\[
\text{type} \\
\text{TNPoss} = T \rightarrow (TN \rightarrow \text{TrnP}) \\
\text{value} \\
\text{free-sector-rule}: TF \times TF \rightarrow TNPoss \\
\text{free-sector-rule}(tf)(r) \equiv \\
\quad \text{if } tf = \square \text{ then } r \text{ else }
\]
let $t : T \cdot t \in \text{dom} \; \text{tf} \wedge \text{smallest}(t)(\text{tf})$ in

let $((hs,ls),\text{trnposs}) = \text{tf}(t)$ in

if $\text{trnposs} = []$ then free_sector_rule($\text{tf}\{t\})(r)$ else

let $tn : TN \cdot tn \in \text{dom} \; \text{trnposs}$ in

if exists_follow_Sect($\text{trnposs}(tn))(hs,ls) \wedge \sim \text{has_free_Sector}(tn,(hs,ls),t)(\text{tf})$

then

let $r' = \text{if } t \in \text{dom} \; r \text{ then } r \text{ else } r \cup \{ t \mapsto [] \}$ end in

free_sector_rule($\text{tf}[t \mapsto ((hs,ls),\text{trnposs}\{tn\})])$

$(r'[t \mapsto r(t) \cup [tn \mapsto \text{trnposs}(tn)]])$ end

else

free_sector_rule($\text{tf}[t \mapsto ((hs,ls),\text{trnposs}\{\text{trn}\})])$($r$)

end end end end end end

\text{smallest}(t)(\text{tf}) \equiv \sim \exists \, t' : T \cdot t' \in \text{dom} \; \text{tf} \wedge t' < t \; \text{pre } t \in \text{dom} \; \text{tf}
Why Current Requirements Engineering is Fundamentally Wrong!

Stakeholder Identification
Domain Acquisition & Analysis
Rough Sketching & Terminology
First Rough Validation

Domain Theory Formation
Domain Engineering

Domain Modelling
- Business Processes
- Intrinsics
- Support Technologies
- Mgt. & Org.
- Rules & Regulations
  - Scripts, Lics., Cntrs.
- Human Behaviour
- Consolidation

Domain Validation
Domain Verification

Requirements Engineering

Requirements Engineering
- Stakeholder Identification
- Domain Acquisition & Analysis
- Rough Sketching & Terminology
- First Rough Validation

Domain Requirements
- Projection
- Instantiation
- Determination
- Extension
- Fitting
- Consolidation

Interface Requirements
- Shared Entities
- Shared Operations
- Shared Events
- Shared Behaviours

Machine Requirements
- Performance
- Dependabilities
- Platform
- Maintenance
- Documentation

Requirements Verification
Requirements Validation
Requirements Feasibility
Requirements Satisfiability
Script Languages [Contract Languages]

• By a domain **script** language we mean
  ★ the definition of a set of licenses and actions
  ★ where these licenses when issued
  ★ and actions when performed have morally obliging power.

• By a domain **contract** language
  ★ a domain script language whose licenses and actions have legally
    binding power,
  ★ that is, their issuance and their invocation may be contested in a
    court of law.
**A Script Language**

- Some common, visual forms of bus timetables are shown in Fig. 4.1.

![Some bus timetables: Spain, India and Norway](image)

**Figure 4.1:** Some bus timetables: Spain, India and Norway
Example 13: Narrative Syntax of a Bus Timetable Script Language

18. Time is a concept covered earlier. Bus lines and bus rides have unique names (across any set of time tables). Hub and link identifiers, HI, LI, were treated from the very beginning.

19. A TimeTable associates to Bus Line Identifiers a set of Journies.

20. Journies are designated by a pair of a BusRoute and a set of BusRides.

21. A BusRoute is a triple of the Bus Stop of origin, a list of zero, one or more intermediate Bus Stops and a destination Bus Stop.

22. A set of BusRides associates, to each of a number of Bus Identifiers a Bus Schedule.

23. A Bus Schedule a triple of the initial departure Time, a list of zero, one or more intermediate bus stop Times and a destination arrival Time.

24. A Bus Stop (i.e., its position) is a Fraction of the distance along a link (identified by a Link Identifier) from an identified hub to an identified hub.

25. A Fraction is a Real properly between 0 and 1.

26. The Journies must be well-formed in the context of some net.
Example 14: Formal Syntax of a Bus Timetable Script Language

type
18. T, BLId, BId
19. TT = BLId \( \mathbb{m} \) Journies
20. Journies' = BusRoute \( \times \) BusRides
21. BusRoute = BusStop \( \times \) BusStop* \( \times \) BusStop
22. BusRides = BId \( \mathbb{m} \) BusSched
23. BusSched = T \( \times \) T* \( \times \) T
24. BusStop == mkBS\( \langle \text{s_fhi:HI, s_ol:LI, s_f:Frac, s_thi:HI} \rangle \)
25. Frac = \{ |r:\text{Real}\cdot0<r<1| \} 
26. Journies = \{ |j:Journies' \exists n:N \cdot \text{wf_Journies}(j)(n) | \}
Example 15: Semantics of a Bus Timetable Script Language

\[
\text{type} \\
\quad \text{Bus} \\
\text{value} \\
\quad \text{obs}_X: \text{Bus} \rightarrow X \\
\text{type} \\
\quad \text{BusTraffic} = T \xrightarrow{m} (N \times (\text{BusNo} \xrightarrow{m} (\text{Bus} \times \text{BPos})))) \\
\quad \text{BPos} = \text{atHub} | \text{onLnk} | \text{atBS} \\
\quad \text{atHub} == \text{mkAtHub}(s_f:L_\text{f}:L_\text{i},s_h:L_\text{hi}:H_\text{i},s_t:L_\text{tl}:L_\text{i}) \\
\quad \text{onLnk} == \text{mkOnLnk}(s_{fhi}:H_\text{fhi}:H_\text{i},s_{ol}:L_\text{i},s_f:F_\text{rac},s_{thi}:H_\text{i}) \\
\quad \text{atBS} == \text{mkAtBS}(s_{fhi}:H_\text{fhi}:H_\text{i},s_{ol}:L_\text{i},s_f:F_\text{rac},s_{thi}:H_\text{i}) \\
\quad \text{Frac} = \{ |r: \text{Real} \cdot 0 < r < 1 | \} \\
\text{value} \\
\quad \text{gen_BusTraffic}: \text{TT} \rightarrow \text{BusTraffic} - \text{infset} \\
\quad \text{gen_BusTraffic}(tt) \text{ as btrfs} \\
\quad \quad \text{post} \forall \text{btrf:BusTraffic} \cdot \text{btrf} \in \text{btrfs} \Rightarrow \text{on\_time(btrf)}(tt)
\]
A Contract Language

• We shall, as for the timetable script, just hint at a contract language.

Example 16: Informal Syntax of Bus Transport Contracts

• An example contract can be ‘schematised’:

```
   con_id: contractor corn contracts contractee ceen
to perform operations "start","cancel","insert","subcontract"
with respect to bus timetable tt.
```
Example 17: Formal Syntax of a Bus Transport Contracts

type
  CId, CNm
  Contract = CId × CNm × CNm × Body
  Body = Op-set × TT
  Op == "conduct" | "cancel" | "insert" | "subcontract"

an example contract:

  (cid,cor,cee,{"start","cancel","insert","subcontract"},tt)
Example 18: Informal Syntax of a Bus Transport Actions

- Example actions can be schematised:
  
  (a) cid: **start bus ride** (blid,bid) **at time** t
  
  (b) cid: **cancel bus ride** (blid,bid) **at time** t
  
  (c) cid: **insert bus ride like** (blid,bid) **at time** t
  
- The schematised license (Slide 78) shown earlier is almost like an action; here is the action form:
  
  (d) cid: **contractee cee is granted a license** cid'

  to perform operations \{"start","cancel","insert",subcontract"\}

  with respect to timetable tt'.
Example 19: Formal Syntax of a Bus Transport Actions

type
Action = CNm × CId × (SubLic | SmpAct) × Time
SmpAct = Start | Cancel | Insert
DoRide == mkSta(s_blid:BLId,s_bid:BId)
Cancel == mkCan(s_blid:BLId,s_bid:BId)
Insert = mkIns(s_blid:BLId,s_bid:BId)
SubCon == mkCon(s_cid:ConId,s_cee:CNm,s_body:(s_ops:Op-set,s_tt:TT))

examples:
(a) (cee,cid,mkRid(blid,id),t)
(b) (cee,cid,mkCan(blid,id),t)
(c) (cee,cid,mkIns(blid,id),t)
(d) (cee,cid,mkCon(cid′,{"start","cancel","insert","subcontract"},tt′),t))

where: cid′ = generate_ConId(cid,cee,t)
Example 20: Semantics of a Bus Transport Contract Language: States

type
Body = Op-set × TT
ConΣ = RcvConΣ × SubConΣ × CorBusΣ
RcvConΣ = CNm \( \xrightarrow{m} \)(CId \( \xrightarrow{m} \)(Body × TT))
SubConΣ = CNm \( \xrightarrow{m} \)(CId \( \xrightarrow{m} \)Body)
BusNo
BusΣ = FreeBusesΣ × ActvBusesΣ × BusHistsΣ
FreeBusesΣ = BusStop \( \xrightarrow{m} \) BusNo-set
ActvBusesΣ = BusNo \( \xrightarrow{m} \) BusInfo
BusInfo = BLId × BId × CId × CNm × BusTrace
BusHistsΣ = Bno \( \xrightarrow{m} \) BusInfo*
BusTrace = (Time × BusStop)*
CorBusΣ = CNm \( \xrightarrow{m} \) (CId \( \xrightarrow{m} \)((BLId × BId) \( \xrightarrow{m} \)(BNo × BusTrace)))
AllBs = CNm \( \xrightarrow{m} \) BusNo-set
Example 21: Semantics of a Bus Transport Contract Language: Constants and Functions

value

cns:\text{CNm-set}, \text{busnos:BNo-set}, \text{ib}\sigma:\text{IB}\Sigma s=\text{CNm }\text{→Bus}\Sigma ,
rcor,icee:\text{CNm} \cdot \text{rcor} \not\in \text{cns}\land \text{icee} \in \text{cns}, \text{itr:BusTraffic},
rcid:\text{ConId}, \text{iops:Op-set}=\{"\text{subcontract}\}\}, \text{itt:TT}, \text{t}_0:\text{Time}
\text{allbs:AllBs} \cdot \text{dom}\text{ allbs}=\text{cns} \cup \{\text{rcor}\}\land \text{rng}\text{ allbs}=\text{busnos},
\text{icon:Contract}=(\text{rcid},\text{rcor},\text{icee},(\text{iops},\text{itt})),
\text{ic}\sigma:\text{Con}\Sigma =([\text{icee} \mapsto \cdots [\text{rcid} \mapsto [\text{icee} \mapsto \text{icon }] ] ] ]
\text{dom}\text{ allbs}=\text{cns} \cup \{\text{rcor}\}\land \text{rng}\text{ allbs}=\text{busnos},
\text{ic}\sigma:\text{Con}\Sigma =([\text{icee} \mapsto \cdots [\text{rcid} \mapsto [\text{icee} \mapsto \text{icon }] ] ] ]
\text{system}: \text{Unit} \rightarrow \text{Unit}
\text{system}() \equiv
\text{cntrctldr}(\text{icee})(\text{il}\sigma(\text{icee}),\text{ib}\sigma(\text{icee}))
\|\{(\|\{\text{cntrctldr}(\text{cee})(\text{il}\sigma(\text{cee}),\text{ib}\sigma(\text{cee}))|\text{cee}\cdot\text{cee} \in \text{cns}\ \setminus \{\text{icee}\}\})
\|\{(\|\{\text{bus\_ride}(\text{b},\text{cee})(\text{rcor},"\text{nil}\))
\|\{\text{bus\_ride}(\text{b},\text{cee})(\text{rcor},"\text{nil}\))
\|\{\text{bus\_ride}(\text{b},\text{cee})(\text{rcor},"\text{nil}\))
\|\{\text{bus\_ride}(\text{b},\text{cee})(\text{rcor},"\text{nil}\))
\|\{\text{bus\_ride}(\text{b},\text{cee})(\text{rcor},"\text{nil}\))
\|\{\text{bus\_ride}(\text{b},\text{cee})(\text{rcor},"\text{nil}\))
\|\{\text{bus\_ride}(\text{b},\text{cee})(\text{rcor},"\text{nil}\))
\|\{\text{bus\_ride}(\text{b},\text{cee})(\text{rcor},"\text{nil}\))
\|\{\text{bus\_ride}(\text{b},\text{cee})(\text{rcor},"\text{nil}\))
\|\{\text{bus\_ride}(\text{b},\text{cee})(\text{rcor},"\text{nil}\))
\|\{\text{bus\_ride}(\text{b},\text{cee})(\text{rcor},"\text{nil}\))
\|\{\text{bus\_ride}(\text{b},\text{cee})(\text{rcor},"\text{nil}\))
\|\{\text{bus\_ride}(\text{b},\text{cee})(\text{rcor},"\text{nil}\))
\|\{\text{bus\_ride}(\text{b},\text{cee})(\text{rcor},"\text{nil}\))
\|\{\text{bus\_ride}(\text{b},\text{cee})(\text{rcor},"\text{nil}\))
\|\{\text{bus\_ride}(\text{b},\text{cee})(\text{rcor},"\text{nil}\))
\|\{\text{bus\_ride}(\text{b},\text{cee})(\text{rcor},"\text{nil}\))
\|\{\text{bus\_ride}(\text{b},\text{cee})(\text{rcor},"\text{nil}\))
\|\{\text{bus\_ride}(\text{b},\text{cee})(\text{rcor},"\text{nil}\))
\|\{\text{bus\_ride}(\text{b},\text{cee})(\text{rcor},"\text{nil}\))
\|\{\text{bus\_ride}(\text{b},\text{cee})(\text{rcor},"\text{nil}\))
\|\{\text{bus\_ride}(\text{b},\text{cee})(\text{rcor},"\text{nil}\))
\|\{\text{bus\_ride}(\text{b},\text{cee})(\text{rcor},"\text{nil}\))
\|\{\text{bus\_ride}(\text{b},\text{cee})(\text{rcor},"\text{nil}\))
\|\{\text{bus\_ride}(\text{b},\text{cee})(\text{rcor},"\text{nil}\))
\|\{\text{bus\_ride}(\text{b},\text{cee})(\text{rcor},"\text{nil}\))
\|\{\text{bus\_ride}(\text{b},\text{cee})(\text{rcor},"\text{nil}\))
The thin lines of Fig. 4.2 denote communication “channels”.

Figure 4.2: An organisation
The Rôle of Domain Engineering in Software Development

**Domain Engineering**
- Stakeholder Identification
- Domain Acquisition & Analysis
- Rough Sketching & Terminology
- First Rough Validation

**Domain Modelling**
- Business Processes
- Intrinsics
- Support Technologies
- Mgt. & Org.
- Rules & Regulations
- Scripts, Lics., Cntrs.
- **Human Behaviour**
- Consolidation
- Domain Verification
- Domain Validation
- Domain Theory Formation

**Requirements Engineering**
- Stakeholder Identification
- Domain Acquisition & Analysis
- Rough Sketching & Terminology
- First Rough Validation

**Domain Requirements**
- Projection
- Instantiation
- Determination
- Extension
- Fitting
- Consolidation

**Interface Requirements**
- Shared Entities
- Shared Operations
- Shared Events
- Shared Behaviours

**Machine Requirements**
- Performance
- Dependabilities
- Platform
- Maintenance
- Documentation

- Requirements Verification
- Requirements Validation
- Requirements Feasibility
- Requirements Satisfiability

---

April 2, 2009, 12:06,

© Dines Bjørner 2008, Fredsvej 11, DK–2840 Holte, Denmark
Human Behaviour

• By human behaviour we mean any of a quality spectrum of carrying out assigned work:
  ★ from careful, diligent and accurate,
  via
  ★ sloppy dispatch, and
  ★ delinquent work,
  to
  ★ outright criminal pursuit.
Example 22: A Diligent Operation

- The `int_Insert` operation of Slide 33

  - was expressed without stating necessary pre-conditions:

27. The `int_Insert` operation takes an `Insert` command and a net and yields either a new net or `chaos` for the case where the insertion command “is at odds” with, that is, is not semantically well-formed with respect to the net.

28. We characterise the “is not at odds”, i.e., is semantically well-formed, that is: `pre_int_Insert(op)(hs,ls)`, as follows: it is a propositional function which applies to `Insert` actions, `op`, and nets, `(hs,ls)`, and yields a truth value if the below relation between the command arguments and the net is satisfied.

Let `(hs,ls)` be a value of type `N`.

29. If the command is of the form `2oldH(hi′,l,hi′)` then

   - `1 hi′` must be the identifier of a hub in `hs`,

   - `2 l` must not be in `ls` and its identifier must (also) not be observable in `ls`, and

   - `3 hi′′` must be the identifier of a(nother) hub in `hs`.

30. If the command is of the form `1oldH1newH(hi,l,h)` then

   - `1 hi` must be the identifier of a hub in `hs`,
2. \( l \) must not be in \( ls \) and its identifier must (also) not be observable in \( ls \), and
3. \( h \) must not be in \( hs \) and its identifier must (also) not be observable in \( hs \).

31. If the command is of the form \( 2\text{newH}(h',l,h'') \) then

- \( 1 \) \( h' \) — left to the reader as an exercise (see formalisation!),
- \( 2 \) \( l \) — left to the reader as an exercise (see formalisation!), and
- \( 3 \) \( h'' \) — left to the reader as an exercise (see formalisation!).

\[\text{value}\]

\[\begin{align*}
28' & \quad \text{pre_int_Insert: Ins} \rightarrow \mathbb{N} \rightarrow \mathbb{B} \\
28'' & \quad \text{pre_int_Insert(Ins(op))}(hs,ls) \equiv \\
29 & \quad \text{case } \text{op} \text{ of } \\
29 & \quad 2\text{oldH}(hi',l,hi'') \rightarrow \{hi',hi''\} \subseteq \text{iohs}(hs), \\
30 & \quad 1\text{oldH1newH}(hi,l,h) \rightarrow \text{hi} \in \text{iohs}(hs) \land \text{hi} \notin \text{hs} \land \text{obs_HI(h)} \notin \text{iohs}(hs), \\
31 & \quad 2\text{newH}(h',l,h'') \rightarrow \{h',h''\} \cap \text{hs} = \{\} \land \{\text{obs_HI(h'),obs_HI(h'')}\} \cap \text{iohs}(hs) = \{\}
\end{align*}\]

- These must be \textbf{carefully} expressed and adhered to
- in order for staff to be said to carry out the link insertion operation \textbf{accurately}. 

\[\text{© Dines Bjørner 2008, Fredsvæj 11, DK–2840 Holte, Denmark}\]

April 2, 2009, 12:06
Example 23: A Sloppy via Delinquent to Criminal Operation

- We replace systematic checks (\(\land\)) with partial checks (\(\lor\)), etcetera,
- and obtain various degrees of sloppy to delinquent, or even criminal behaviour.

\[
\begin{align*}
28' & \text{pre\_int\_Insert: } \text{Ins} \to N \to \text{Bool} \\
28'' & \text{pre\_int\_Insert(Ins(op))(hs,ls)} \equiv \\
\ast 2 & s\_l(op) \not\in \text{ls} \land \text{obs\_LI(s\_l(op))} \not\in \text{iols(l)} \land \\
\text{case}\ \text{op}\ \text{of} \\
29 & \text{oldH(hi',l,hi'') } \rightarrow \text{hi'} \in \text{iohs(hs)} \lor \text{hi''isin iohs(hs)}, \\
30 & \text{oldH1newH(hi,l,h)} \rightarrow \text{hi} \in \text{iohs(hs)} \lor \text{h} \not\in \text{hs} \lor \text{obs\_HI(h)} \not\in \text{iohs(hs)}, \\
31 & \text{newH(h',l,h'')} \rightarrow \{h',h''\} \cap \text{hs} = \{} \lor \{\text{obs\_HI(h')},\text{obs\_HI(h'')}\} \cap \text{iohs(hs)} = \{}
\end{align*}
\]
Dialectics

• So now you should have a practical and technical “feel” for domain engineering:
  ✭ What it takes to express a domain model.

• But there is lots’ more: We have not shown you
  ✭ (i) the rôle of domain stakeholders:
    † (i.1) how to identify them,
    † (i.2) how to involve them and
    † (i.3) how they help validate resulting domain descriptions.
  ✭ (ii) the domain (ii.1) knowledge acquisition and (ii.2) analysis processes,
  ✭ (ii) the domain (ii.1) model verification and (ii.2) validation and
    processes, and
  ✭ (iii) the domain theory R&D process.
Can we agree that we cannot,
* as professional software engineers,
* start on gathering requirements,
* let alone prescribing these
* before we have understood the domain?

Can we agree that, “ideally”, we must therefore
* first R&D the domain model
* before we can embark on any requirements prescription process?

By “ideally” we mean the following:
* Ideally domain engineering should fully precede requirements engineering,
* but for many practical reasons we must co-develop domain descriptions
  “hand-in-hand” with requirements prescriptions.
* And that is certainly feasible, when done with care.
* So we shall, for years assume this to be the case.
Pragmatics

- While the software industry “humps along”:
  - co-developing domain descriptions and requirements
  - with their clients, or, for COTS, with their marketing departments,

- private and public research centres should and will embark on
  - large scale (5–8 manyears/year),
  - long range projects (5–8 year)
  - foundational research and development (R&D) of infrastructure component domain models of
the financial service industry:
  ◦ banking (all forms);
  ◦ insurance (all forms);
  ◦ portfolio management;
  ◦ securities trading:
    ○ brokers,
    ○ traders,
  ◦ commodities and
  ◦ stock etc. exchanges;

transportation:
  ◦ road,
  ◦ rail,

healthcare:
  ◦ physicians,
  ◦ hospitals,
  ◦ clinics,
  ◦ pharmacies, etc.;

“the market”:
  ◦ consumers,
  ◦ retailers,
  ◦ wholesalers, and
  ◦ the supply chain;

etcetera.
PSI’09: 15–19 June 2009 and: Why Current Requirements Engineering is Fundamentally Wrong!
Requirements Engineering

• We cannot possibly,
  ★ within the confines of a seminar talk
  ★ and a reasonably sized paper
• cover, however superficially,
  ★ both informal
  ★ and formal
examples of requirements engineering.
• Instead we shall just briefly mention the major stages and sub-stages of requirements modeling:

★ Domain Requirements: those which can be expressed solely using terms from the domain description;

★ Interface Requirements: those which can be expressed using terms both from the domain description and from IT; and

★ Machine Requirements: those which can be expressed solely using terms from IT.

________________________  IEEE Definition of Requirements  __________________

★ By IT requirements we understand (cf. IEEE Standard 610.12):
   ◊ “A condition or capability needed by a user to solve a problem or achieve an objective on a computing machine”.

• By computing machine we shall understand a, or the, combination of computer (etc.) hardware and software that is the target for, or result of the required computing systems development.
The Role of Domain Engineering in Software Development, PSI'09: 15–19 June 2009

Domain Engineering
- Stakeholder Identification
- Domain Acquisition & Analysis
- Rough Sketching & Terminology
- First Rough Validation

Domain Modelling
- Business Processes
- Intrinsics
- Support Technologies
- Mgt. & Org.
- Rules & Regulations
- Scripts, Lics., Contracts
- Human Behaviour
- Consolidation
- Domain Verification
- Domain Validation
- Domain Theory Formation

Requirements Engineering
- Stakeholder Identification
- Domain Acquisition & Analysis
- Rough Sketching & Terminology
- First Rough Validation

Domain Requirements
- Projection
- Instantiation
- Determination
- Extension
- Fitting
- Consolidation

Interface Requirements
- Shared Entities
- Shared Operations
- Shared Events
- Shared Behaviours

Machine Requirements
- Performance
- Dependabilities
- Platform
- Maintenance
- Documentation

Requirements Verification

Requirements Validation

Requirements Feasibility

Requirements Satisfiability

Software Design
- IT System Design
- Software Architecture Design
- Component Design
- Module Design
- Code Design

© Dines Bjørner 2008, Fredsvej 11, DK–2840 Holte, Denmark
By *domain requirements* we mean such which can be expressed solely using terms from the domain description.

To construct the domain requirements:

- the domain engineer
- together with the various groups of requirements stakeholder

“apply” the following “domain-to-requirements” operations to a copy of the domain description:

- *projection*,
- *instantiation*,
- *determination*,
- *extension* and *fitting*.

First we briefly characterise these.
The Domain-to-Requirements Operations

- The ‘domain-to-requirements’ operations cannot be automated.
- They increasingly “turn” the copy of the domain description into a domain requirements prescription.

★ Projection removes all the domain phenomena and concepts for which the customer does not need IT support.
★ Instantiation makes a number of entities: simple, operations, events and behaviours, less abstract, more concrete.
★ Determination makes the emerging requirements entities more determinate.
★ Extension introduces new, computable entities that were not possible in the non-IT domain.
★ Fitting merges the domain requirements prescription with those of other IT developments.
Domain Engineering
- Stakeholder Identification
- Domain Acquisition & Analysis
- Rough Sketching & Terminology
- First Rough Validation
- Domain Theory Formation

Requirements Engineering
- Stakeholder Identification
- Domain Acquisition & Analysis
- Rough Sketching & Terminology
- First Rough Validation

Domain Modelling
- Business Processes
- Intrinsics
- Support Technologies
- Mgt. & Org.
- Rules & Regulations
- Scripts, Lics., Contracts
- Human Behaviour
- Consolidation
- Domain Verification
- Domain Validation

Domain Requirements
- Projection
- Instantiation
- Determination
- Extension
- Fitting
- Consolidation
- Interface Requirements
  - Shared Entities
  - Shared Operations
  - Shared Events
  - Shared Behaviours
- Machine Requirements
  - Performance
  - Dependabilities
  - Platform
  - Maintenance
  - Documentation

Requirements Verification
- Requirements Validation
- Requirements Feasibility
- Requirements Satisfiability

Software Design
- IT System Design
- Software Architecture Design
- Component Design
- Module Design
- Code Design

© Dines Bjørner 2008, Fredsvej 11, DK–2840 Holte, Denmark
By *interface requirements* we mean such which those which can be expressed using terms from both the domain description and from IT, that is, terminology of hardware and of software.

- When phenomena and concepts of the domain are also to be represented by the machine,
  - these phenomena and concepts are said to be *shared* between the domain and the machine;
  - the requirements therefore need be expressed both
    - in terms of phenomena and concepts of the domain and
    - in terms of phenomena and concepts of the machine.
Shared Phenomena and Concepts

• A shared phenomenon or concept is either
  ★ a simple entity, ★ an event or
  ★ an operation, ★ a behaviour.

• **Shared simple entities** need
  ★ to be initially input to the machine and
  ★ their machine representation need to be
  ★ regularly, perhaps real-time refreshed.

• **Shared operations** need
  ★ to be interactively performed by
  ★ human or other agents of the domain
  ★ and by the machine.
Shared events are shared in the sense that

- their occurrence in the domain
- must be made known to the machine.

Shared behaviours need

- to occur in the domain and in the machine
- by alternating means,
- that is, a protocol need be devised.

For each of these four kinds of interface requirements

- the reqs. engineers work with the reqs. stakeholders
- to determine the properties of these forms of sharing.

These interface requirements are then narrated and formalised.

They are always “anchored” in specific items of the domain description.
Why Current Requirements Engineering is Fundamentally Wrong!

Domain Engineering
- Stakeholder Identification
- Domain Acquisition & Analysis
- Rough Sketching & Terminology
- First Rough Validation

Domain Modelling
- Business Processes
- Intrinsics
- Support Technologies
- Mgt. & Org.
- Rules & Regulations
- Scripts, Lics., Contracts
- Human Behaviour
- Consolidation

Domain Requirements
- Projection
- Instantiation
- Determination
- Extension
- Fitting
- Consolidation

Interface Requirements
- Shared Entities
- Shared Operations
- Shared Events
- Shared Behaviours

Machine Requirements
- Performance
- Dependabilities
- Platform
- Maintenance
- Documentation

Domain Theory Formation

Requirements Engineering
- Stakeholder Identification
- Domain Acquisition & Analysis
- Rough Sketching & Terminology
- First Rough Validation

Requirements Verification

Software Design
- IT System Design
- Software Architecture Design
- Component Design
- Module Design
- Code Design

© Dines Bjørner 2008, Fredsvej 11, DK–2840 Holte, Denmark
Machine Requirements

- By *machine requirements*
  - we mean those which can be expressed
  - solely using terms from the machine,
  - that is, terminology of hardware and of software.

- We shall not cover any principles or techniques for developing machine requirements,
- but shall just list the very many issues that must be captured by a machine requirements.
• Performance
  ✷ Storage
  ✷ Time
  ✷ Software Size
• Dependability
  ✷ Accessibility
  ✷ Availability
  ✷ Reliability
  ✷ Robustness
  ✷ Safety
  ✷ Security
• Maintenance
  ✷ Adaptive
  ✷ Corrective
  ✷ Perfective
  ✷ Preventive
• Platform (P)
  ✷ Development P
  ✷ Demonstration P
  ✷ Execution P
  ✷ Maintenance P
• Documentation
  Requirements
• Other Requirements

• The machine requirements are usually not so easily, formalised, if at all, with today’s specification language tools.

• Extra great care must therefore be exerted in their narration.

• Some formal modelling calculations, like fault (tree) analysis, can be made in order to justify quantitative requirements.
Why “Current” Requirements Engineering (RE) is Flawed

- Current, conventional RE has no scientific basis:
  - “My” RE starts with a domain model.
  - It provides the scientific basis.
  - “Derivation” of domain and interface requirement provides a further scientific basis.

- The separation of concerns:
  - domain model, in-and-by-itself, and
  - the requirements projection, instantiation, determination, extension and fitting operators

  provide a basis for scientific analysis.

- Current, conventional RE does not have these bases.

- Current, research into and practice of conventional RE “must be stopped”
  - if we are to pursue Software Engineering in a professionally responsible manner.
Conclusion

Summary — A Wrap Up

• We have illustrated the triptych concept:
  ★ from domains via requirements to software.
• We spent most time on domain engineering.
• We just sketched major requirements engineering concepts.
• And we assumed you know how to turn formal requirements into correct software designs!
Conclusion

Dialectics

• So, are we clear on this:
  ★ (i) that we must understand the domain before we express the requirements;
  ★ (ii) that we can “derive” major parts of the requirements prescription from the domain description;
  ★ (iii) that domains are far more “stable” than requirements;
  ★ (iv) that prescribing requirements with no prior domain description is thoroughly unsound;
  ★ (v) that describing [prescribing] domains [requirements] both informally (narratives) and formally (formal specifications) helps significantly towards consistent specifications; and
  ★ (vi) that we must therefore embrace the triptych:
  ★ from domains via requirements to software.
Implication: Theory-work

• So, get on with it!
• Pick up one or another of the new
  ★ domain engineering ideas:
    ◊ business processes, ◊ domain theories,
    ◊ facets, ◊ etc.,
  or the new
  ★ requirements engineering ideas:
    ◊ projection, ◊ determination, ◊ fitting,
    ◊ instantiation, ◊ extension and

• research them, write papers about it.
Implication: Engineering-work — Extrovert Applications

• But do it in connection with real life, actual domains:

  ★ banking,          ★ hospitalisation,          ★ container line
  ★ insurance,        ★ bus & tax            ★ shipping,
  ★ stock exchange and        ★ transport,         ★ etcetera.
     brokerage,
     ★ rail transport,

• That is, “build” some impressive domain theories!
Implication: Engineering-work — Introspective Applications

- By introspective applications we mean such as providing software for, or such as
  - the Internet,
  - the Web,
  - operating systems
  - database management,
  - data communication,
  - etcetera, etcetera,

- Also these are lack proper domain descriptions.
For More on Domain and Requirements Engineering

- For details on domain and requirements engineering we refer to:

  * **Software Engineering:**
and the upcoming book:

*Software Engineering,*

* Vol. I: The Triptych Approach,
* Vol. II: A Model Development.

To be submitted to Springer for evaluation, expected published 2009.

- This book (draft) is the basis for lectures at

  * Techn. Univ. of Graz, Austria,  Nov.-Dec. 2008;
  * Univ. of Saarland, Germany,  March 2009; and
  * Univ. of Tokyo, Japan,  Fall (Oct.-Nov.) 2009.
Conclusion

For More on Extrovert Applications

We refer to some indicative Internet-based reports:

- air traffic
  [www.imm.dtu.dk/~db/brisbane.pdf](http://www.imm.dtu.dk/~db/brisbane.pdf) and [airtraffic.pdf](http://www.imm.dtu.dk/~db/airtraffic.pdf);

- container line industry:
  [www.imm.dtu.dk/~db/container-paper.pdf](http://www.imm.dtu.dk/~db/container-paper.pdf);

- the ‘Market’:
  [www.imm.dtu.dk/~db/themarket.pdf](http://www.imm.dtu.dk/~db/themarket.pdf);

- IT security:
  [www.imm.dtu.dk/~db/5lectures/it-system-security-ISO.pdf](http://www.imm.dtu.dk/~db/5lectures/it-system-security-ISO.pdf);

- oil industry:
  Appendix: [www.imm.dtu.dk/~db/de-p.pdf](http://www.imm.dtu.dk/~db/de-p.pdf);

- railways:
  [www.railwaydomain.org/](http://www.railwaydomain.org/);

- transportation (in general):
  Appendix: [www.imm.dtu.dk/~db/tseb.pdf](http://www.imm.dtu.dk/~db/tseb.pdf);
• etcetera.
Conclusion

Introvert Applications
Software Engineering Archeology

• In general I would prefer to see precise domain models of
  • the Internet,
  • ‘Cloud Computing’,
  • the Web,
  • Windows Vista,
  • Linux and
  • idealised SQL

• as the basis for
  • requirements and
  • software

• that claim that they are “based” on
  • the Internet,
  • ‘Cloud Computing’,
  • the Web,
  • Windows Vista,
  • Linux and/or
  • SQL.

• Here is clearly a fascination engineering task.

• I see the Internet as an instantiation of ‘Cloud Computing’.
For More on Research Topics

- A number of research topics of domain theory has been outlined in:
Acknowledgements

• Thanks for inviting me to PSI’09.
• Indeed, very many THANKS.