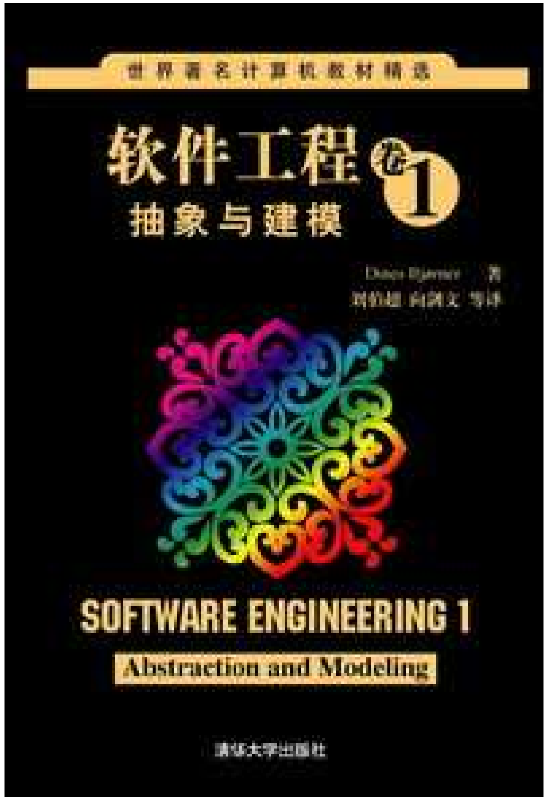

FINAL LAST HAUL !

Begin of Lecture 9: Last Session — Conclusion

Comparisons and What Have We Achieved

FM 2012 Tutorial, Dines Bjørner, Paris, 28 August 2012

Tutorial Schedule

- **Lectures 1–2** 9:00–9:40 + 9:50–10:30

1 **Introduction** Slides 1–35

2 **Endurant Entities: Parts** Slides 36–110

- **Lectures 3–5** 11:00–11:15 + 11:20–11:45 + 11:50–12:30

3 **Endurant Entities: Materials, States** Slides 111–142

4 **Perdurant Entities: Actions and Events** Slides 143–174

5 **Perdurant Entities: Behaviours** Slides 175–285

Lunch **12:30–14:00**

- **Lectures 6–7** 14:00–14:40 + 14:50–15:30

6 **A Calculus: Analysers, Parts and Materials** Slides 286–339

7 **A Calculus: Function Signatures and Laws** Slides 340–377

- **Lectures 8–9** 16:00–16:40 + 16:50–17:30

8 **Domain and Interface Requirements** Slides 378–424

√ 9 **Conclusion: Comparison to Other Work** **Slides 428–460**

√ **Conclusion: What Have We Achieved** **Slides 425–427 + 461–472**

13. Conclusion

- This document,
  - meant as the basis for my tutorial
  - at **FM 2012** (**CNAM**, Paris, August 28),
  - "grew" from a paper being written for possible journal publication.
    - Sections 2–3 possibly represent two publishable journal papers.
    - Section 4 has been "added" to the 'tutorial' notes.

- The style of the two tutorial “parts”,
  - Sects. 2–3 and
  - Sect. 4
  - are, necessarily, different:
    - Sects. 2–3
      are in the form of research notes,
    - whereas Sect. 4
      is in the form of “lecture notes” on methodology.
  - Be that as it may. Just so that you are properly notified !

13.1. Comparison to Other Work

- In this section we shall only compare
  - ◈ our contribution to domain engineering as presented in the section on domain entities
  - ◈ to that found in the broader literature with respect to the software engineering term 'domain'.
- We shall not compare
  - ◈ our contribution to requirements engineering
  - ◈ as surveyed in the section on requirements engineering.
  - ◈ to that, also, found in the broader requirements engineering literature.
- Finally we shall also not compare
  - ◈ our work on a description calculus
  - ◈ as we find no comparable literature!

13.1.1. Ontological Engineering:

- Ontological engineering is described mostly on the Internet, see however [Benjamins+Fensel98].
- Ontology engineers build ontologies.
- And ontologies are, in the tradition of ontological engineering, *"formal representations of a set of concepts within a domain and the relationships between those concepts"* — expressed usually in some logic.
- Published ontologies usually consists of thousands of logical expressions.
- These are represented in some, for example, low-level mechanisable form so that they can be interchanged between ontology groups building upon one-anothers work and processed by various tools.

- There does not seem to be a concern for "deriving" such ontologies into requirements for software.
- Usually ontology presentations
  - either start with the presentation
  - or makes reference to its reliance

  of an upper ontology.
- Instead the ontology databases
  - appear to be used for the computerised
  - discovery and analysis
  - of relations between ontologies.

- The `TripTych` form of domain science & engineering differs from conventional **ontological engineering** in the following, essential ways:
  - The `TripTych` domain descriptions rely essentially on a "built-in" **upper ontology**:
    - types, abstract as well as model-oriented (i.e., concrete) and
    - actions, events and behaviours.
  - Domain science & engineering is not, to a first degree, concerned with modalities, and hence do not focus on the modelling of
    - knowledge and belief,
    - necessity and possibility, i.e., alethic modalities,
    - epistemic modality (certainty),
    - promise and obligation (deontic modalities),
    - etcetera.

13.1.2. Knowledge and Knowledge Engineering:

- The concept of knowledge has occupied philosophers since Plato.
  - No common agreement on what 'knowledge' is has been reached.
  - From Wikipedia we may learn that
    - *knowledge is a familiarity with someone or something;*
      - *it can include facts, information, descriptions, or skills acquired through experience or education;*
      - *it can refer to the theoretical or practical understanding of a subject;*
    - *knowledge is produced by socio-cognitive aggregates*
      - *(mainly humans)*
      - *and is structured according to our understanding of how human reasoning and logic works.*

- The aim of **knowledge engineering** was formulated, in 1983, by an originator of the concept, Edward A. Feigenbaum [Feigenbaum83]:
  - **knowledge engineering** is an engineering discipline
  - that involves integrating knowledge into computer systems
  - in order to solve complex problems
  - normally requiring a high level of human expertise.

- **Knowledge engineering** focuses on
  - continually building up (acquire) large, shared data bases (i.e., **knowledge bases**),
  - their continued maintenance,
  - testing the validity of the stored 'knowledge',
  - continued experiments with respect to **knowledge representation**,
  - etcetera.

- Knowledge engineering can, perhaps, best be understood in contrast to algorithmic engineering:
  - In the latter we seek more-or-less conventional, usually imperative programming language expressions of algorithms
    - whose algorithmic structure embodies the knowledge
    - required to solve the problem being solved by the algorithm.
  - The former seeks to solve problems based on an interpreter inferring possible solutions from logical data. This logical data has three parts:
    - a collection that "mimics" the semantics of, say, the imperative programming language,
    - a collection that formulates the problem, and
    - a collection that constitutes the knowledge particular to the problem.
- We refer to [BjornerNilsson1992].

- The concerns of `TripTych` domain science & engineering is based on that of algorithmic engineering.
  - Domain science & engineering is not aimed at
    - letting the computer solve problems based on
    - the knowledge it may have stored.
  - Instead it builds models based on knowledge of the domain.
- Further references to seminal exposés of **knowledge engineering** are [Studer1998,Kendal2007].

13.1.3. Domain Analysis:

- There are different "schools of domain analysis".
  - **Domain analysis**, or **product line analysis** (see below), as it was first conceived in the early 1980s by James Neighbors
    - is the analysis of related software systems in a domain
    - to find their common and variable parts.
    - It is a model of wider business context for the system.
  - This form of domain analysis turns matters "upside-down":
    - it is the set of software "systems" (or packages)
    - that is subject to some form of inquiry,
    - albeit having some domain in mind,
    - in order to find common features of the software
    - that can be said to represent a named domain.

- In this section we shall mainly be comparing the TripTych approach to domain analysis to that of Reubén Prieto-Dĩaz's approach [Prieto-Diaz:1987,Prieto-Diaz:1990,Prieto-Diaz:1991].
- Firstly, the two meanings of **domain analysis** basically coincide.
- Secondly, in, for example, [Prieto-Diaz:1987], Prieto-Dĩaz's domain analysis is focused on the very important stages that precede the kind of **domain modelling** that we have described:
  - major concerns are
    - selection of what appears to be similar, but specific entities,
    - identification of common features,
    - abstraction of entities and
    - classification.
  - **Selection** and **identification** is assumed in our approach, but we suggest to follow the ideas of Prieto-Dĩaz.
  - **Abstraction** (from values to types and signatures) and **classification** into parts, materials, actions, events and behaviours is what we have focused on.

- All-in-all we find Prieto-Dĩaz's work very relevant to our work:
  - relating to it by providing guidance to pre-modelling steps,
  - thereby emphasising issues that are necessarily informal,
  - yet difficult to get started on by most software engineers.
- Where we might differ is on the following:
  - although Prieto-Dĩaz does mention a need for **domain specific languages**,
  - he does not show examples of **domain descriptions** in such `DSL`s.
  - We, of course, basically use mathematics as the `DSL`.
- In the `TripTych` approach to **domain analysis**
  - we provide a full ontology — cf. Sects. 2.–10. and
  - suggest a **domain description calculus**.
- In our approach
  - we do not consider requirements, let alone software components,
  - as do Prieto-Dĩaz,

  but we find that that is not an important issue.

13.1.4. Software Product Line Engineering:

- Software product line engineering, earlier known as domain engineering,
  - is the entire process of **reusing domain knowledge** in the production of new software systems.
- Key concerns of **software product line engineering** are
  - **reuse**,
  - the building of repositories of **reusable software components**, and
  - **domain specific languages** with which to, more-or-less automatically build software based on **reusable software components**.

- These are not the primary concerns of TripTych domain science & engineering.
  - But they do become concerns as we move from domain descriptions to requirements prescriptions.
  - But it strongly seems that software product line engineering is not really focused on the concerns of domain description — such as is TripTych domain engineering.
  - It seems that software product line engineering is primarily based, as is, for example, FODA: Feature-oriented Domain Analysis, on analysing features of software systems.
  - Our [dines-maurer] puts the ideas of software product lines and model-oriented software development in the context of the TripTych approach.
- Notable sources on software product line engineering are [dom:Bayer:1999,dom:Weiss:1999,dom:Ardis:2000,dom:Thiel:2000,dom:Ha

13.1.5. Problem Frames:

- The concept of problem frames is covered in [mja2001a].
- Jackson's prescription for software development focuses on the "triple development" of descriptions of
  - the problem world,
  - the requirements and
  - the machine (i.e., the hardware and software) to be built.
- Here domain analysis means, the same as for us, the problem world analysis.

- In the **problem frame** approach the software developer plays three, that is, all the `TripTych` rôles:
  - **domain engineer**,
  - **requirements engineer** and
  - **software engineer**

  "all at the same time",
- well, iterating between these rôles repeatedly.
- So, perhaps belabouring the point,
  - **domain engineering** is done only to the extent needed by the prescription of **requirements** and the **design** of **software**.
- These, really are minor points.

- But in "restricting" oneself to consider
  - only those aspects of the domain which are mandated by the **requirements prescription**
  - and **software design**

  one is considering a potentially smaller fragment [Jackson2010Facs] of the domain than is suggested by the **TripTych** approach.
- At the same time one is, however, sure to
  - consider aspects of the domain
  - that might have been overlooked when pursuing **domain description development**
  - the **TripTych**, "more general", approach.

13.1.6. Domain Specific Software Architectures (DSSA):

- It seems that the concept of DSSA
  - was formulated by a group of ARPA[30] project "seekers"
  - who also performed a year long study (from around early-mid 1990s);
  - key members of the DSSA project were Will Tracz, Bob Balzer, Rick Hayes-Roth and Richard Platek [dom:Trasz:1994].
- The [dom:Trasz:1994] definition of **domain engineering** is *"the process of creating a DSSA:*
  - ***domain analysis*** and ***domain modelling***
  - *followed by creating a **software architecture***
  - *and populating it with **software components**."*

[30] ARPA: The US DoD Advanced Research Projects Agency

- This definition is basically followed also by [Mettala+Graham:1992,Shaw+Garlan:1996,Medvidovic+Colbert:2004].
- Defined and pursued this way, DSSA appears,
  - notably in these latter references, to start with the
  - with the analysis of software components, "per domain",
  - to identify commonalities within application software,
  - and to then base the idea of software architecture
  - on these findings.

- Thus DSSA turns matter "upside-down" with respect to TripTych requirements development
  - by starting with software components,
  - assuming that these satisfy some requirements,
  - and then suggesting domain specific software
  - built using these components.
- This is not what we are doing:
  - We suggest that requirements
    - can be "derived" systematically from,
    - and related back, formally to domain descriptionss
    - without, in principle, considering software components,
    - whether already existing, or being subsequently developed.

- Of course, given a **domain descriptions**
  - it is obvious that one can develop, from it, any number of **requirements prescriptions**
  - and that these may strongly hint at shared, (to be) implemented **software components**;
- but it may also, as well, be the case
  - two or more **requirements prescriptions**
  - "derived" from the same **domain description**
  - may share no **software components** whatsoever !
- So that puts a "damper" of my "enthusiasm" for DSSA.

- It seems to this author that had the `DSSA` promoters
  - based their studies and practice on also using formal specifications,
  - at all levels of their study and practice,
  - then some very interesting insights might have arisen.

13.1.7. Domain Driven Design (DDD)

- Domain-driven design (DDD)[31]
  - *"is an approach to developing software for complex needs*
  - *by deeply connecting the implementation to an evolving model of the core business concepts;*
  - *the premise of domain-driven design is the following:*
    - *placing the project's primary focus on the core domain and domain logic;*
    - *basing complex designs on a model;*
    - *initiating a creative collaboration between technical and domain experts to iteratively cut ever closer to the conceptual heart of the problem."*[32]

[31] Eric Evans: http://www.domaindrivendesign.org/
[32] http://en.wikipedia.org/wiki/Domain-driven_design

- We have studied some of the DDD literature,
  - mostly only accessible on The Internet, but see also [Haywood2009],
  - and find that it really does not contribute to new insight into **domain**s such as wee see them:
  - it is just "plain, good old software engineering cooked up with a new jargon.

13.1.8. Feature-oriented Domain Analysis (FODA):

- Feature oriented domain analysis (FODA)
  - is a domain analysis method
  - which introduced feature modelling to domain engineering
  - FODA was developed in 1990 following several U.S. Government research projects.
  - Its concepts have been regarded as critically advancing software engineering and software reuse.
- The US Government supported report [KyoKang+et.al.:1990] states: "*FODA is a necessary first step*" for software reuse.

- To the extent that
  - TripTych domain engineering
  - with its subsequent **requirements engineering**

  indeed encourages reuse at all levels:
  - **domain descriptions** and
  - **requirements prescription**,

  we can only agree.
- Another source on FODA is [Czarnecki2000].
- Since FODA "leans" quite heavily on 'Software Product Line Engineering' our remarks in that section, above, apply equally well here.

13.1.9. Unified Modelling Language (UML)

- Three books representative of UML are [Booch98,Rumbaugh98,Jacobson99].
- The term **domain analysis** appears numerous times in these books,
  - yet there is no clear, definitive understanding
  - of whether it, the **domain**, stands for entities in the domain such as we understand it,
  - or whether it is wrought up, as in several of the 'approaches' treated in this section, to wit, Items [3,4,6,7,8], with
    - either **software design** (as it most often is),
    - or **requirements prescription**.

- Certainly, in UML,
  - in [Booch98,Rumbaugh98,Jacobson99] as well as
  - in most published papers claiming "adherence" to UML,
  - that domain analysis usually
    - is manifested in some UML text
    - which "models" some **requirements** facet.
  - Nothing is necessarily wrong with that;
  - but it is therefore not really the TripTych form of **domain analysis**
    - with its concepts of abstract representations of endurant and perdurants, and
    - with its distinctions between **domain** and **requirements**, and
    - with its possibility of "deriving"
      - ∗ **requirements prescriptions** from
      - ∗ **domain descriptions**.

- There is, however, some important notions of UML
  - and that is the notions of
    - class diagrams,
    - objects, etc.
  - How these notions relate to the discovery
    - of part types, unique part identifiers, mereology and attributes, as well as
    - action, event and behaviour signatures and channels,
  - as discovered at a particular domain index,
  - is not yet clear to me.
  - That there must be some relation seems obvious.
- We leave that as an interesting, but not too difficult, research topic.

13.1.10. Requirements Engineering:

- There are in-numerous books and published papers on requirements engineering.
  - A seminal one is [AvanLamsweerde2009].
  - I, myself, find [SorenLauesen2002] full of very useful, non-trivial insight.
  - [Dorfman+Thayer:1997:IEEEComp.Soc.Press] is seminal in that it brings a number or early contributions and views on requirements engineering.

- Conventional text books, notably [Pfleeger2001,Pressman2001,Sommerville2006] all have their "mandatory", yet conventional coverage of **requirements engineering**.
  - None of them "derive" requirements from domain descriptions,
    - yes, OK, from domains,
    - but since their description is not mandated
    - it is unclear what "the domain" is.
  - Most of them repeatedly refer to **domain analysis**
    - but since a written record of that **domain analysis** is not mandated
    - it is unclear what "domain analysis" really amounts to.

- Axel van Laamsweerde's book [AvanLamsweerde2009] is remarkable.
  - Although also it does not mandate descriptions of domains
  - it is quite precise as to the relationships between domains and requirements.
  - Besides, it has a fine treatment of the distinction between **goals** and **requirements**,
  - also formally.
- Most of the advices given in [SorenLauesen2002]
  - can beneficially be followed also in
  - **TripTych requirements development**.
- Neither [AvanLamsweerde2009] nor [SorenLauesen2002] preempts **TripTych requirements development**.

13.1.11. Summary of Comparisons

- It should now be clear from the above that
  - basically only Jackson's *problem frames* really take
    - the same view of **domain**s and,
    - in essence, basically maintain similar relations between
      - **requirements prescription** and
      - **domain description**.
  - So potential sources of, we should claim, mutual inspiration
    - ought be found in one-another's work —
    - with, for example, [ggjz2000,Jackson2010Facs],
    - and the present document,
    - being a good starting point.

13.2. What Have We Achieved and Future Work

- Sect. 13.1 has already touched upon, or implied,
  - a number of 'achievement' points and
  - issues for future work.
- Here is a summary of 'achievement' and future work items.

- We claim that there are three major contributions being reported upon:
  - (i) the separation of **domain engineering** from **requirements engineering**,
  - (ii) the separate treatment of **domain science & engineering**:
    - as "free-standing" with respect, ultimately, to computer science,
    - and endowed with quite a number of **domain analysis principle**s and **domain description principle**s; and
  - (iii) the identification of a number of techniques
    - for "deriving" significant fragments of **requirements prescription**s from **domain description**s —
    - where we consider this whole relation between **domain engineering** and **requirements engineering** to be novel.

- Yes, we really do consider the possibility of a systematic
  - 'derivation' of significant fragments of **requirements prescriptions** from **domain descriptions**
  - to cast a different light on **requirements engineering**.
- What we have not shown in this tutorial is
  - the concept of **domain facets**;
  - this concept is dealt with in [dines:facs:2008] —
  - but more work has to be done to give a firm theoretical understanding of **domain facets** of
    - **domain intrinsics**,
    - **domain support technology**,
    - **domain scripts**,
    - **domain rules and regulations**,
    - **domain management and organisation**, and
    - **human domain**behaviour.

13.3. General Remarks

- Perhaps belaboring the point:
  - one can pursue creating and studying **domain descriptions**
  - without subsequently aiming at **requirements development**,
  - let alone **software design**.
- That is, **domain descriptions**
  - can be seen as
    - "free-standing",
    - of their "own right",
    - useful in simply just understanding
    - domains in which humans act.

- Just like it is deemed useful
  - that we study "Mother Nature",
  - the physical world around us,
  - given before humans "arrived";
- so we think that
  - there should be concerted efforts to study and create **domain models**,
  - for use in
    - studying "our man-made domains of discourses";
    - possibly proving laws about these domains;
    - teaching, from early on, in middle-school, the domains in which the middle-school students are to be surrounded by;
    - etcetera

- How far must one formalise such **domain descriptions** ?
  - Well, enough, so that possible laws can be mathematically proved.
  - Recall that **domain descriptions** usually will or must be developed by **domain researchers** — not necessarily **domain engineers** —
    - in research centres, say universities,
    - where one also studies physics.

- And, when we base **requirements development** on **domain descriptions**,
  - as we indeed advocate,
  - then the **requirements engineers**
  - must understand the formal **domain descriptions**,
  - that is, be able to perform formal
    - domain projection,
    - domain instantiation,
    - domain determination,
    - domain extension,

    etcetera.

- This is similar to the situation in classical engineering
  - which rely on the sciences of physics,
  - and where, for example,
    - *Bernoulli's equations*,
    - *Navier-Stokes equations*,
    - *Maxwell's equations*,
    - etcetera
  - were developed by physicists and mathematicians,
  - but are used, daily, by engineers:
    - read and understood,
    - massaged into further differential equations, etcetera,
    - in order to calculate (predict, determine values), etc.

- Nobody would hire non-skilled labour
  - for the engineering development of airplane designs
    - unless that "labourer" was skilled in *Navier-Stokes equations*,

  or
  - for the design of mobile telephony transmission towers
    - unless that person was skilled in *Maxwell's equations*.

- So we must expect a future, we predict,
  - where a subset of the software engineering candidates from universities
    - are highly skilled in the development of
      - formal **domain descriptions**
      - formal **requirements prescriptions**
  - in at least one domain, such as
    - *transportation*, for example,
      - air traffic,
      - railway systems,
      - road traffic and
      - shipping;

      or
    - *manufacturing*,
    - *services* (health care, public administration, etc.),
    - *financial industries*, or the like.

13.4. Acknowledgements

- I thank the tutorial organisers of the `FM 2012` event for accepting my Dec. 31. 2011 tutorial proposal.
- I thank that part of participants
  - who first met up for this tutorial this morning (Tuesday 28 August, 2012)
  - to have remained in this room for most, if not all of the time.
- I thank colleagues and PhD students around Europe
  - for having listened to previous,
  - somewhat less polished versions of this tutorial.
  - I in particular thank Drs. **Magne Haveraaen** and **Marc Bezem** of the University of Bergen for providing an important step in the development of the present material.
- And I thank my wife
  - for her patience during the spring and summer of 2012
  - where I ought to have been tending to the garden, etc. !

End of Lecture 9: Last Session — Conclusion

Comparisons and What Have We Achieved

FM 2012 Tutorial, Dines Bjørner, Paris, 28 August 2012

THANKS AGAIN — HAVE A NICE CONFERENCE