A New Foundation for Computing Science
A Research & Experimental Engineering Programme

Dines Bjørner
Guimarães, Portugal

Fredsvej 11, DK–2840 Holte, Denmark
September 16, 2015: 09:08

In honour of Prof. José Nuno Oliveira
From Domain via Requirements to Software Design

1.1. The Compiler Development Approach

Figure 1: The Ada Compiler Software Development Graph [Bjø77]
1.2. – as 5 MSc Thesis Projects for 6 Students

Figure 2: [BO80]
1.3. Domain Engineering

1.3.1. Denotational Semantics

Figure 3: McCarthy [McC60, McC62], Strachey & Scott [Str68, Sco70, SS71, Sco72]
1.4. Requirements Engineering

1.4.1. The Landin SECD Machine and Reynolds Closures

Figure 4: Landin [Lan64, Lan65a, Lan65b], Reynolds [Rey70, Rey72]
1.4.2. Macro-Expansion Semantics

Abstract Syntax

Static Semantics

Domain

Higher-order

Resumption

1st Order

Requirement

“Denotational” Semantics

1st Order

Functional Interpreter

Imperative Stack

Bekic

Semantics of an A[da]-Code

Machine Language

Dynamic Semantics

Operational “Parallel” Ada

Run-time Interpreter

Multi-pass Algorithm

Multi-pass Administrator

Back End Pass #1

Back End Pass #2

Back End Pass #3

Back End Pass #4

Run-time Admin.

Run-time Interpreter

Back End

Back End

Back End

Back End

Software Design

Front End Pass #1

Front End Pass #2

Front End Pass #3

Front End Pass #4

Multi-pass Admin.

Back End Pass #1

Back End Pass #2

Back End Pass #3

Back End Pass #4

Run-time Admin.

Multi-pass Administrator

Figure 5: Bekić [Bek84]
1.4.3. Compiling Algorithm

Figure 6: McCarthy & Painter [MP66]
1.4.4. Machine Requirements

Figure 7: The Ada Compiler Software Development Graph
1.5. **Lines of [VDM+comment] Specifications and Man Years**

![Diagram](image)

**Figure 8:** The Ada Compiler Software Development Graph
The Thesis of This Talk

• To describe a Domain is to give semantics to its endurants and perdurants.
  ◦ That is, a Domain is viewed as a language.
  ◦ Description emphasis is put on semantic domains

• To prescribe Requirements is to “derive” these from a domain description.
  ◦ The Requirements are for an interpretive machine.

• To specify a/the Software design is to refine it from the requirements prescription.
• To verify correctness of the software design is to
  ◆ formally test,
  ◆ model check and
  ◆ prove property theorems.
• $D, S \models R$

• $S \models R$ helps ensure correctness.
• $D, S \models R$ helps ensure that product meets client expectations.
The Development Dogma

3.1. The Specification Dogma

• In order to develop *Software*
  we must have a reasonable understanding of the requirements.

• In order to understand the *Requirements*
  we must have a reasonable understanding of the domain.

• In order to understand the *Domain*
  we must analyse & describe it.
3.2. **The Verification Dogma**

- In order to have trust in the *Software*
  it must be related formally to a *Requirements*.

- In order to have trust in the *Requirements*
  it must be related formally to a *Domain description*.
3.3. Domain Engineering

3.3.1. Domain Analysis: Manifest & Non-manifest Phenomena

Figure 9: An Ontology of Manifest & Non-manifest Phenomena
3.3.2. Domain Analysis Prompts

A Triptych Manifest Domain Ontology

Describable

Non-describable

Endurant

Perdurant

Discrete Continuous

Action Event Behaviour

Part Component Material

Atomic Composite

Unique Identification

Mereology Attributes

is_endurant, is_perdurant

is_discrete, is_continuous

has_material, has_components

is_part, is_component, is_material

is_atomic, is_composite

Figure 10: Analysis Prompts
3.3.3. Domain Description Prompts

A Triptych Manifest Domain Ontology

Describable

Endurant

Discrete

Continuous

Part

Component

Material

Composite

Perdurant

Action

Event

Behaviour

Non-describable

Atomic

Unique Identification

Mereology

Attributes

observe_Part_Sorts

observe_Part_Types

observe_Components

observe_Material

observe_Unique_Identifier

observe_Mereology

observe_Attributes

Figure 11: Description Prompts
3.3.4. Domain Analysis: Non-manifest Properties

Figure 12: Attributes Analysis Prompts
3.4. Requirements Engineering

- **Three Stages**
  - Domain Requirements
  - Interface Requirements
  - Machine Requirements

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

- **Interface Requirements**
  - Shared Phenomena
  - Shared Endurants
  - Shared Actions
  - Shared Events
  - Shared Behaviours
So What’s at Stake?

4.1. “States-of-Affairs”

• It seems that compiler development using formal methods
  - such as in the DDC Ada Project (1981–1984)
  - is still not developed the right way in industry
  - and is also not taught that way at very, very many universities.

• It also seems that most other “application software”
  - is mostly not developed properly:
    - from domain descriptions
    - via (therefrom derived) requirements prescriptions
    - to software design etc.
4.2. What Would it Take?

4.2.1. Computer Science

• By computer science we understand the study and knowledge of the artifacts that can exist inside computers.

4.2.2. Computing Science

• By computing science we understand the study and knowledge of how to construct those artifacts.

4.2.3. Formal Method

• By a formal method we understand a set of principles for selecting and applying techniques and tools for constructing an artifact — where the tools and techniques can be formalised, i.e., given a logic/algebraic basis.
4.2.4. **A Remedy**

- This speaker suggests, as far as universities are concerned,
  - that we put more emphasis on **computing science**,  
  - that we do more **research** into and **teach** more **formal methods**,  
  - that we **research** and **teach**  
    - **domain science & engineering** and  
    - **domain, interface & machine requirements.**  
  - and that we  
    - do **experimental research** into  
    - and **pathfinder develop**  
    - **domains** and **domain applications.**
4.3. Justification

- The Dansk Datamatik Centers Ada Compiler project demonstrated that using formal methods can lead to trustworthy software:
  Less than 3% of original resources spent on corrective, perfective and adaptive maintenance since 1984.
- So for programming languages we know how to do it.
- But for application domain categories such as government systems: taxation, policing, social services, etc. we repeatedly hear of “IT scandals”.
- I am sure that many of the abstractions, concepts and ideas of programming languages and interpreter/compiler development can form a strong basis for domain science & engineering.
Relevant Publications & Reports

• [Bjø16b, 2015] is the definitive paper on Manifest Domains: Analysis & Description

• [Bjø16a, 2015] is the definitive paper on From Domain Descriptions to Requirements Prescriptions – A Different Approach to Requirements Engineering
5.1. Further Domain Science & Engineering Papers

• Web page www.imm.dtu.dk/~dibj/domains/ lists the published papers and reports mentioned below.

• I have thought about domain engineering for more than 25 years.

• But serious, focused writing only started to appear as from [Bjø06, Part IV] — with [Bjø03, Bjø97] being exceptions:
  ◊ [Bjø07, 2007] suggests a number of domain science and engineering research topics;
  ◊ [Bjø10a, 2008] covers the concept of domain facets;
  ◊ [BE10, 2008] explores compositionality and Galois connections.
[Bjø08, Bjø10b, 2008,2009] show how to systematically, but, of course, not automatically, “derive” requirements prescriptions from domain descriptions;

[Bjø11a, 2008] takes the triptych software development as a basis for outlining principles for believable software management;


[Bjø11b, 2010] presents, based on the TripTych view of software development as ideally proceeding from domain description via requirements prescription to software design, concepts such as software demos and simulators;
5. Relevant Publications & Reports 5.1. Further Domain Science & Engineering Papers


- the first part of [Bjø14b, 2014] is a precursor for [Bjø16b, 2015] with its second part presenting a first formal model of the elicitation process of analysis and description based on the prompts more definitively presented in the current paper; and

- [Bjø14c, 2014] focus on domain safety criticality.
5.2. Some Domain Descriptions

5.2.1. 1990s: UNU–IIST

1 Scheduling and Rescheduling of Trains (China)  
[BGP95, BGH+97]

2 Ministry of Finance (Vietnam)  
[DCT+96] and [VGJM02, Chapter 5]

3 Radio/Telecommunications System (The Philippines)  
[DG96, LM97] and [VGJM02, Chapter 4]

4 Airlines (Vietnam) [AM96]

5 Manufacturing: Production Processes [VGJM02, Chapter 7]

6 Travel Planning [VGJM02, Chapter 8]

7 Enterprise Management [JA97]
5.2.2. 2000s and on ...

8 A Railway Systems Domain

   it-security.pdf (2006)

10 A Container Line Industry Domain

11 The “Market”:
   Consumers, Retailers, Wholesalers, Producers
   themarket.pdf (2007)
12 What is Logistics?
   logistics.pdf (2009)

13 A Domain Model of Oil Pipelines
   pipeline.pdf (2009)

14 Transport Systems
   comet/comet1.pdf (2010)

15 The Tokyo Stock Exchange

16 On Development of Web-based Software. A Divertimento
   wfdftp.pdf (2010)

17 Documents (incomplete draft)
Conclusion

• So, welcome to a wonderful world of
  Domain Science & Engineering!

• What is there to wait for!?

• Bring your Computing/Computer Science group up to speed!

• Your students will love it.

• Young researchers will thrive.
7. References


