A New Foundation for Computing Science

A Research & Experimental Engineering Programme

Dines Bjørner

Pörtschach am Wörthersee, Kärnten, Österreich

Fredsvej 11, DK–2840 Holte, Denmark

5–7 October 2015

Programmiersprachen und Grundlagen der Programmierung

From Domain via Requirements to Software Design

1.1. The Compiler Development Approach

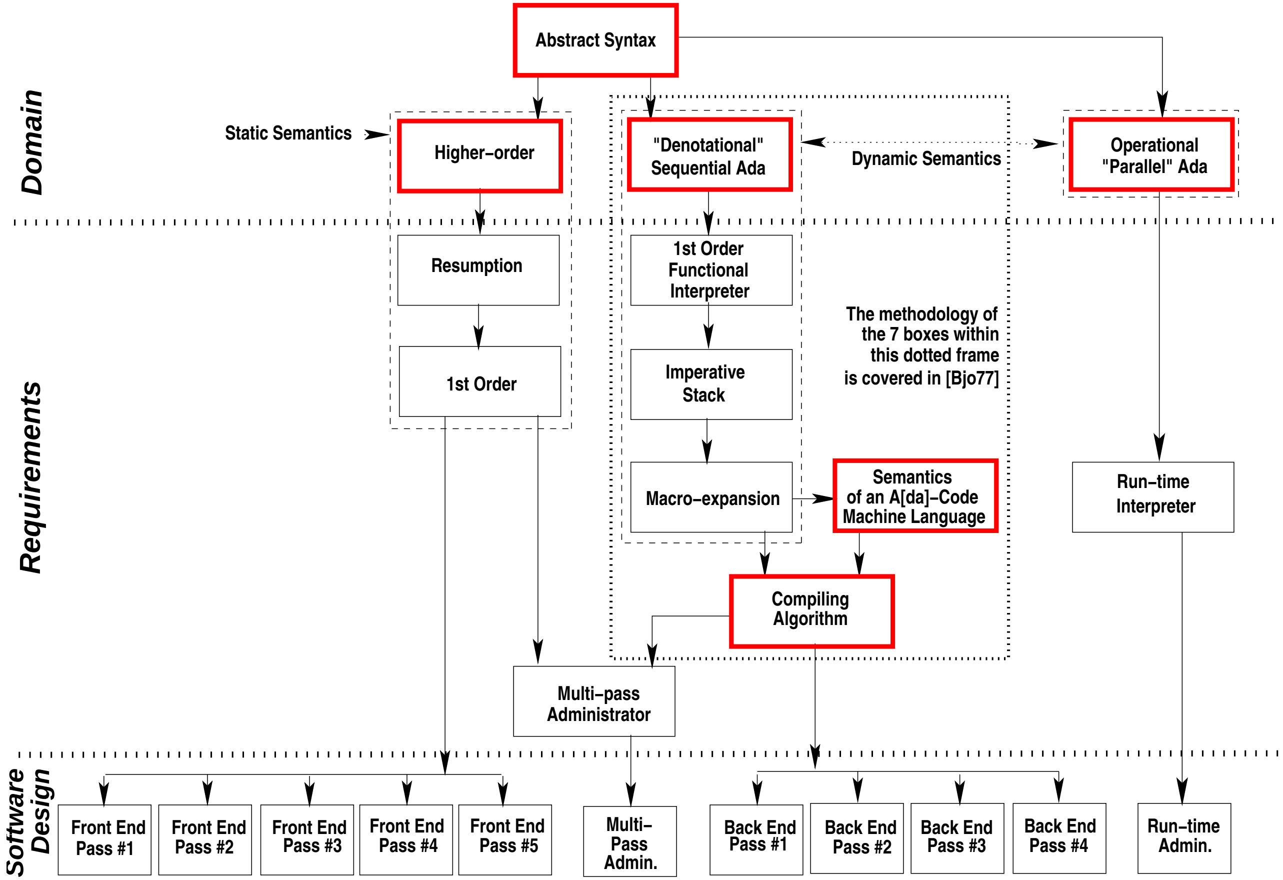

Figure 1: The Ada Compiler Software Development Graph [Bjø77]

1.2. – as 5 MSc Thesis Projects for 6 Students

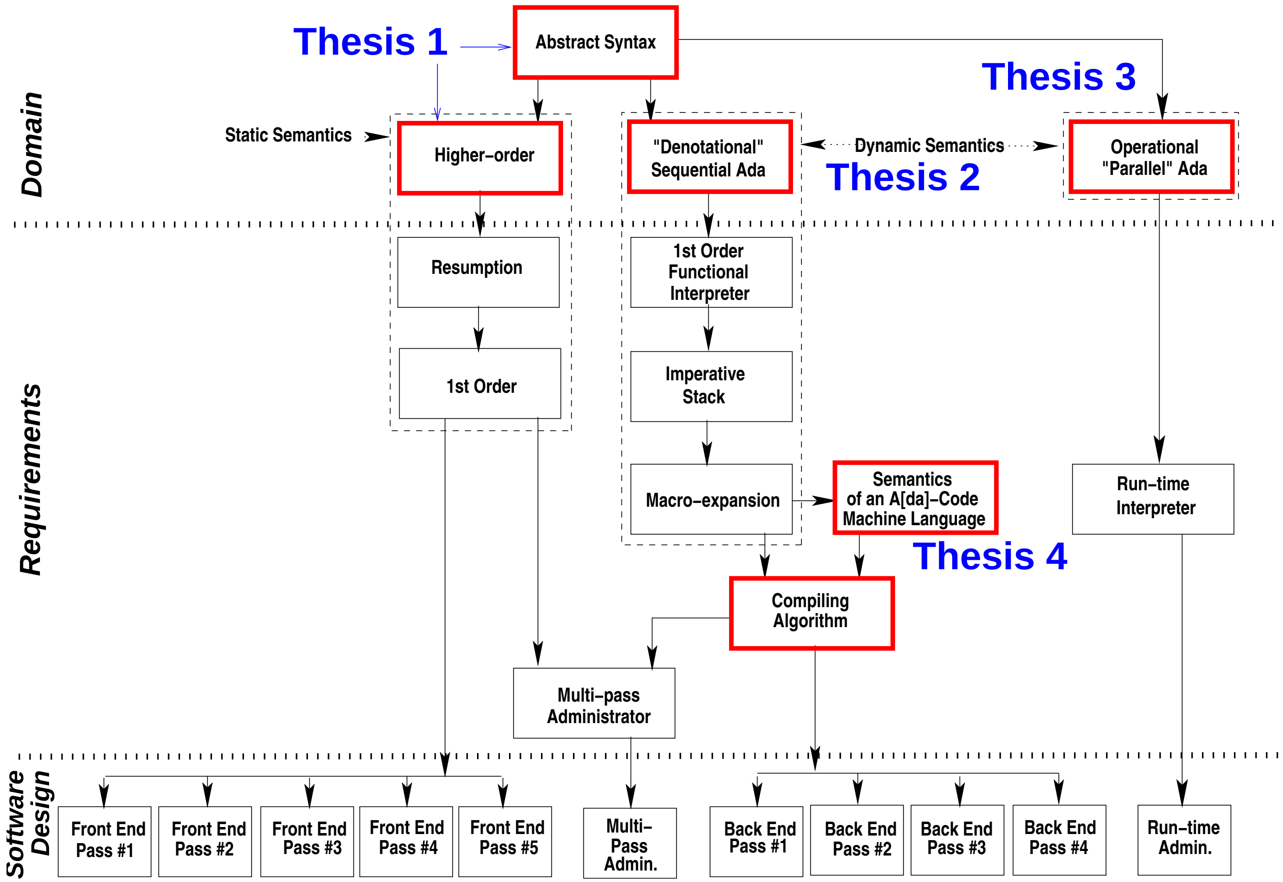

Figure 2: [BO80]

1.3. Domain Engineering

1.3.1. Denotational Semantics

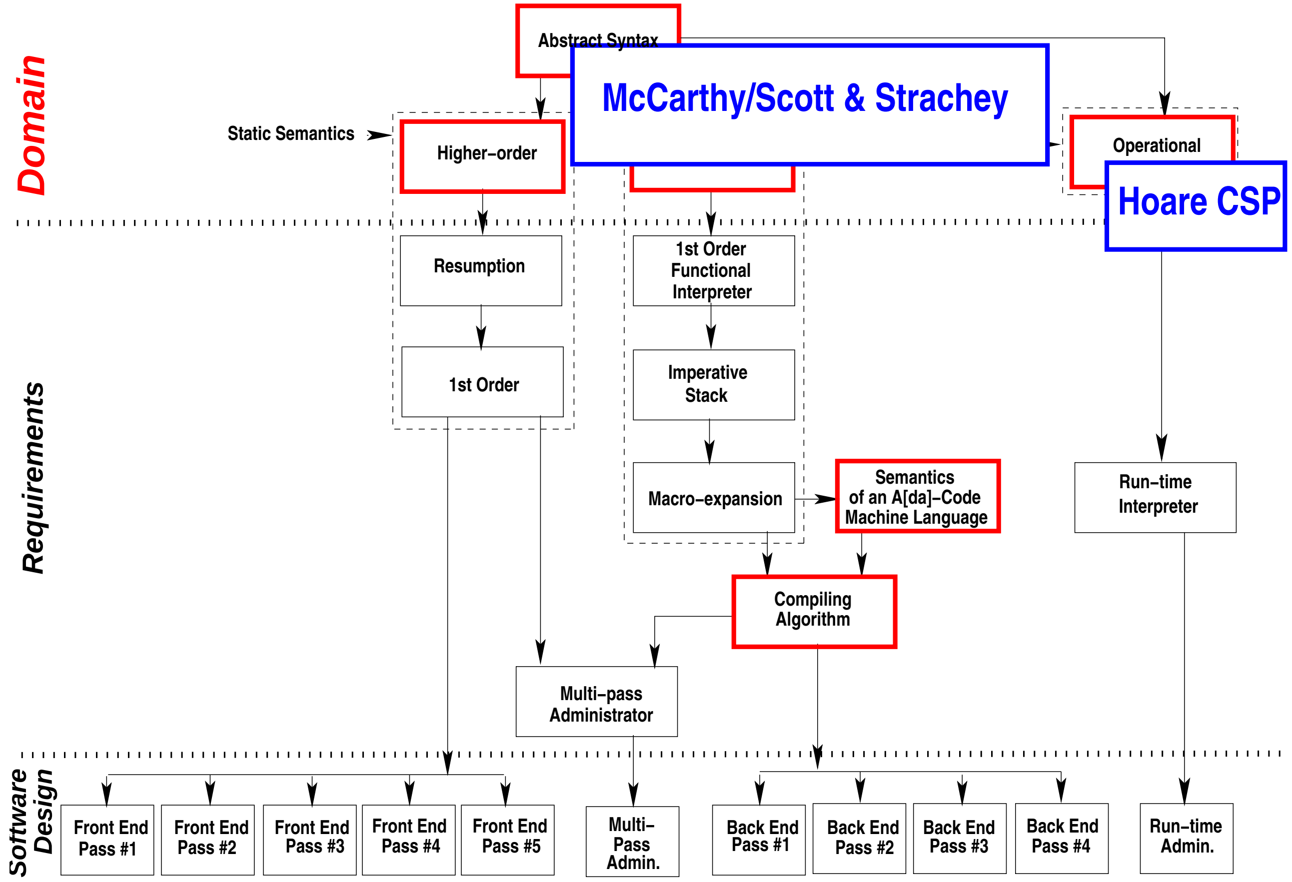

Figure 3: McCarthy [McC60, McC62], Strachey & Scott [Str68, Sco70, SS71, Sco72]

1.4. Requirements Engineering

1.4.1. The Landin SECD Machine and Reynolds Closures

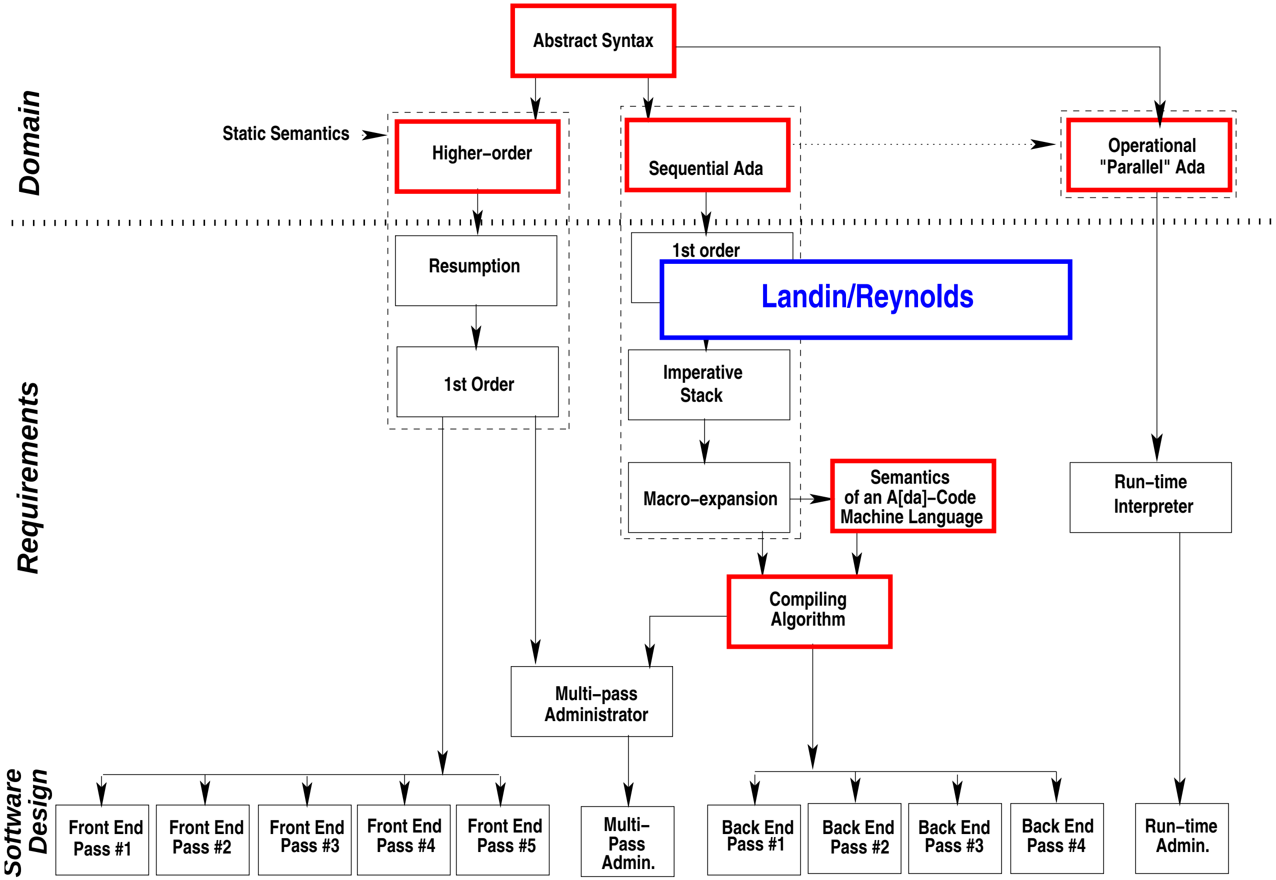

Figure 4: Landin [Lan64, Lan65a, Lan65b], Reynolds [Rey70, Rey72]

1.4.2. Macro-Expansion Semantics

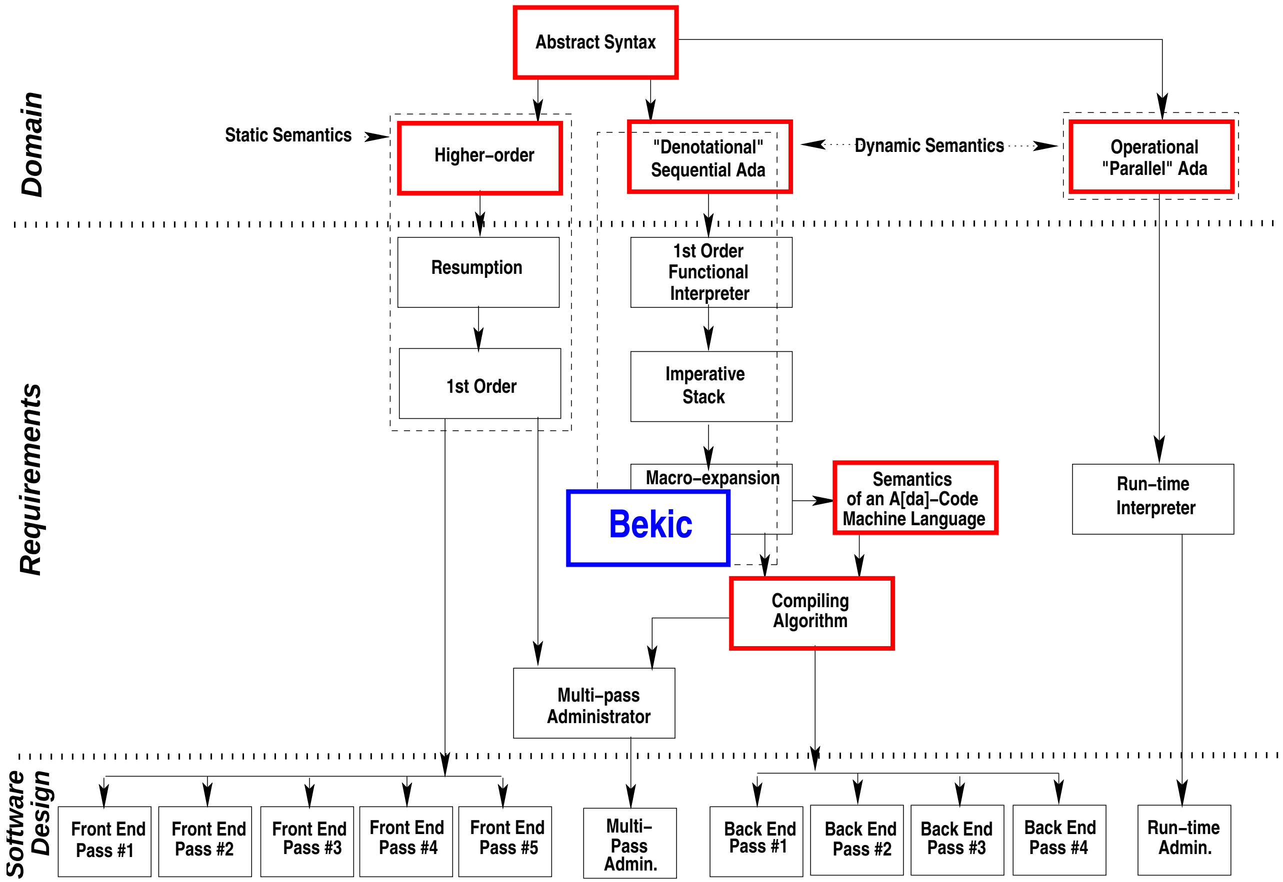

Figure 5: Bekič [Bek84]

1.4.3. Compiling Algorithm

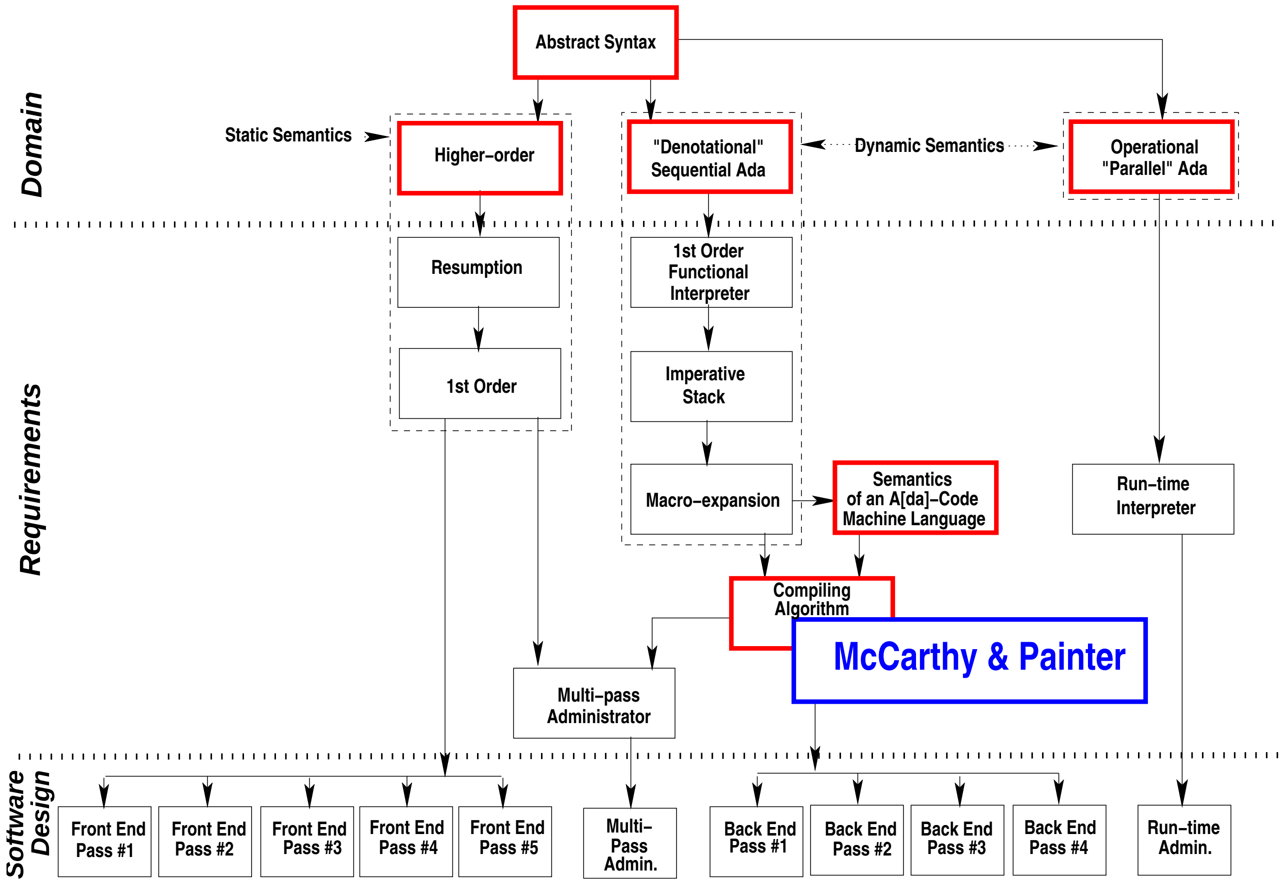

Figure 6: McCarthy & Painter [MP66]

1.4.4. Machine Requirements

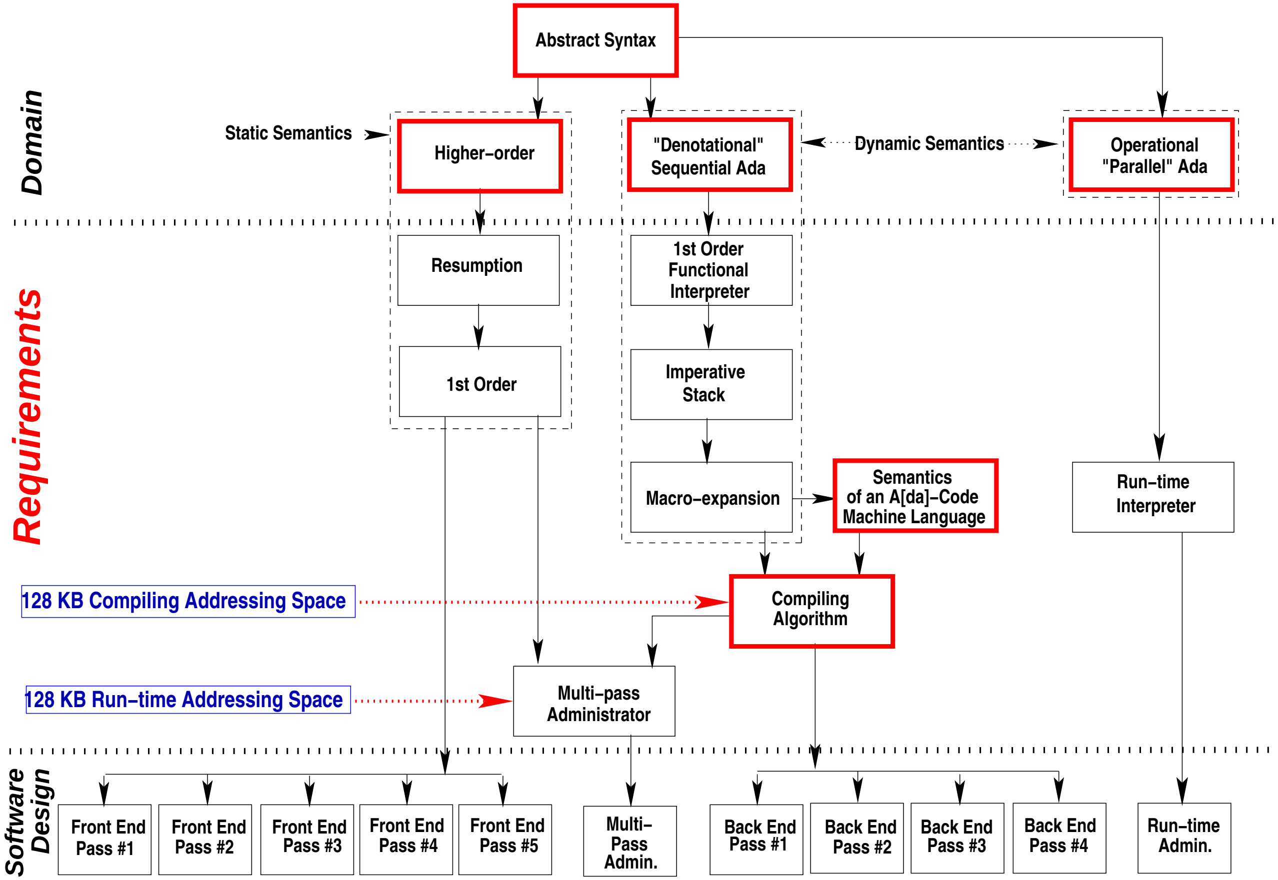

Figure 7: The Ada Compiler Software Development Graph

1.5. Lines of [VDM+comment] Specifications and Man Years

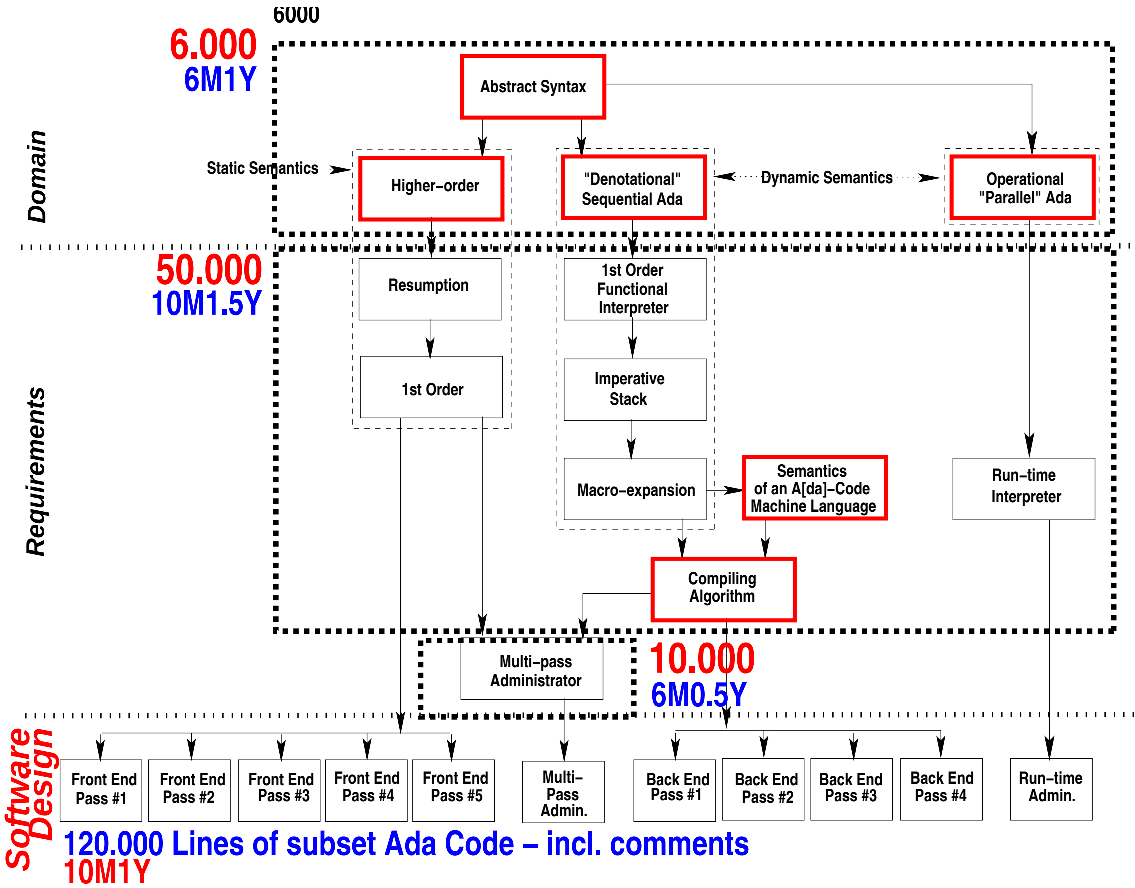

Figure 8: The Ada Compiler Software Development Graph

The Thesis of This Talk

- To describe a $\mathcal{D}$omain is to give semantics to its endurants and perdurants.
  - That is, a $\mathcal{D}$omain is viewed as a language.
  - Description emphasis is put on semantic domains
- To prescribe $\mathcal{R}$equirements is to "derive" these from a domain description.
  - The $\mathcal{R}$equirements are for an interpretive machine.
- To specify a/the $\mathcal{S}$oftware 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.
- $\mathcal{D},\mathcal{S} \models \mathcal{R}$
- $\mathcal{S} \models \mathcal{R}$ helps ensure correctness.
- $\mathcal{D},\mathcal{S} \models \mathcal{R}$ helps ensure that product meets client expectations.

The Development Dogma
3.1. The Specification Dogma

- In order to develop $\mathcal{S}$oftware
  we must have a reasonable understanding of the requirements.

- In order to understand the $\mathcal{R}$equirements
  we must have a reasonable understanding of the domain.

- In order to understand the $\mathcal{D}$omain
  we must analyse & describe it.

3.2. The Verification Dogma

- In order to have trust in the $\mathcal{S}$oftware
  it must be related formally to a $\mathcal{R}$equirements.

- In order to have trust in the $\mathcal{R}$equirements
  it must be related formally to a $\mathcal{D}$omain description.

3.3. Domain Engineering

3.3.1. Domain Analysis: Manifest & Non-manifest Phenomena

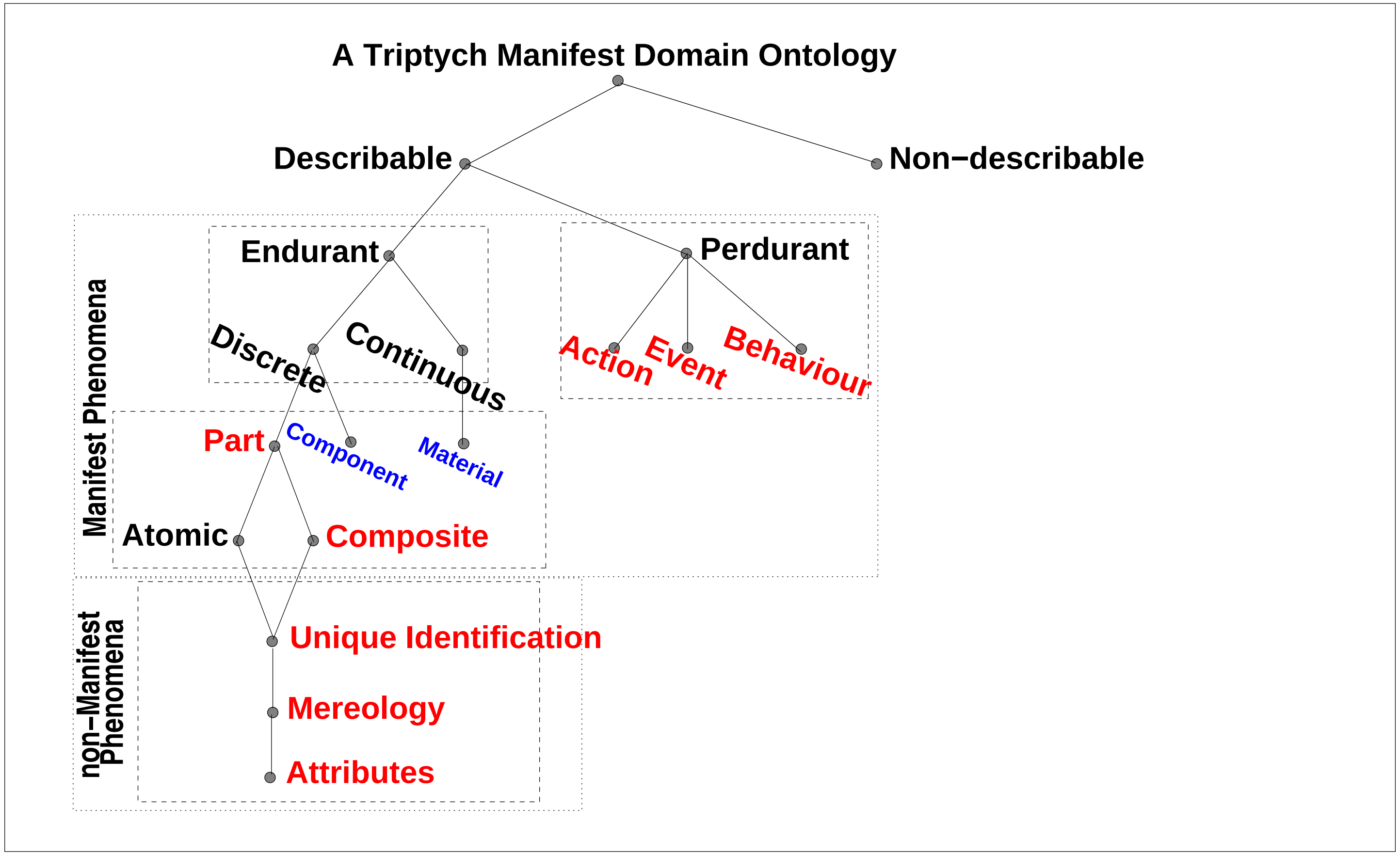

Figure 9: An **Ontology** of Manifest & Non-manifest Phenomena

3.3.2. Domain Analysis Prompts

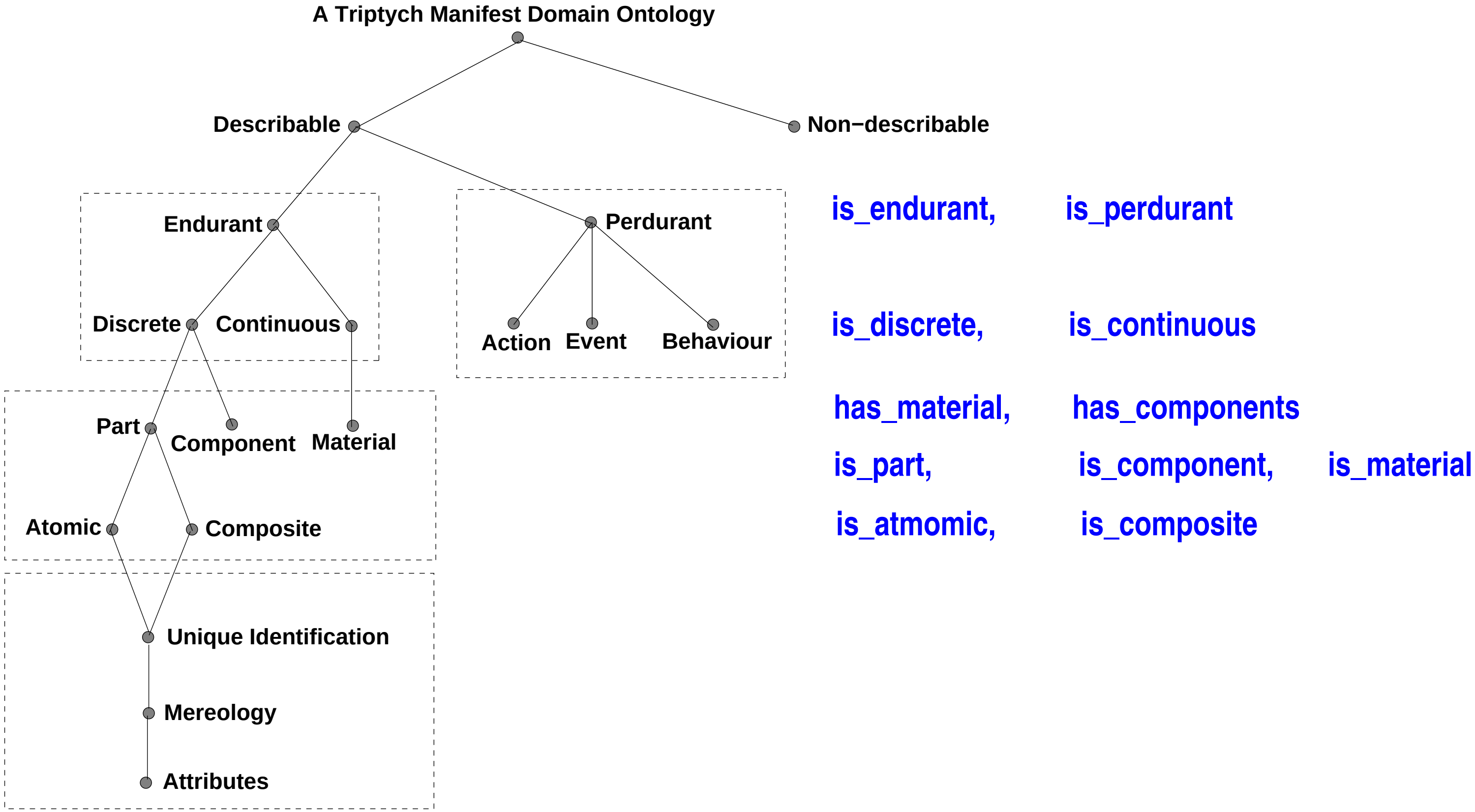

Figure 10: Analysis Prompts

3.3.3. Domain Description Prompts

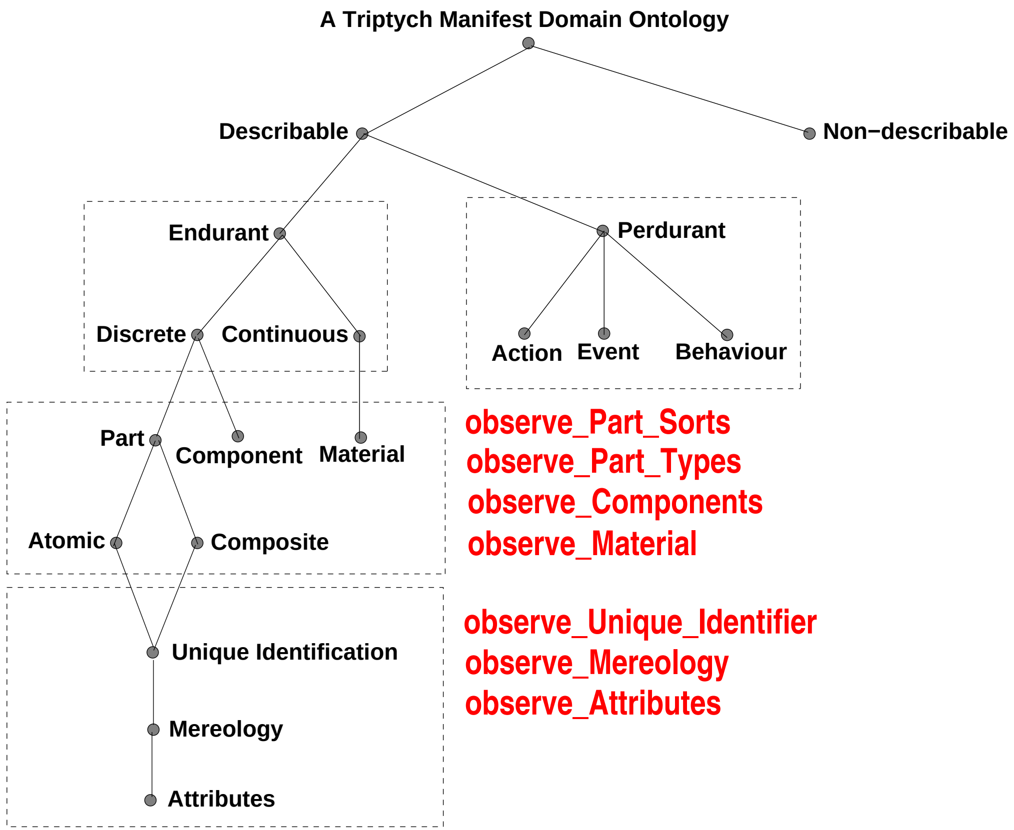

Figure 11: Description Prompts

3.3.4. Domain Analysis: Non-manifest Properties

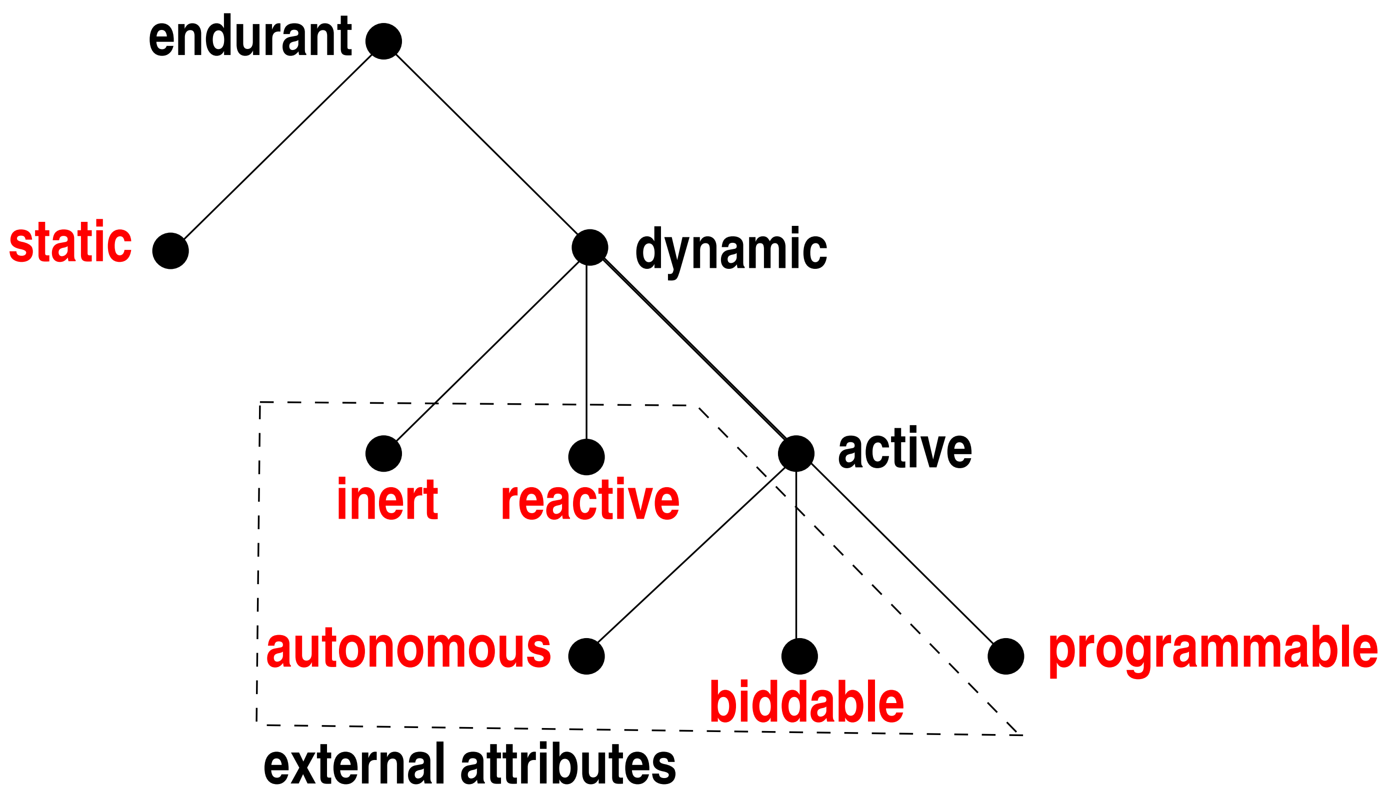

Figure 12: Attributes Analysis Prompts

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;**
- **[Bjø09, Bjø14a, 2009,2013]** presents a model for **Stanisław Leśniewski's** [CV99] concept of **mereology;**
- **[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;**

- **[Bjø13, 2012]** analyses the `TripTych`, especially its domain engineering, approach, with respect to **Maslow's Theory of Human Motivation.** Psychological Review 50(4) (1943):370-96; and **Motivation and Personality,** (Third Edition, Harper and Row Publishers, 1954.) and Peterson's and Seligman's **Character strengths and virtues: A handbook and classification.** (Oxford University Press, 2004);
- 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**
`http://euler.fd.cvut.cz/railwaydomain/` (2003)

9 **Models of IT Security. Security Rules & Regulations**
`it-security.pdf` (2006)

10 **A Container Line Industry Domain**
`container-paper.pdf` (2007)

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**
`todai/tse-1.pdf` and `todai/tse-2.pdf` (2010)

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

17 **Documents (incomplete draft)**
`doc-p.pdf` (2013)

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

[AM96] Dao Nam Anh and Richard Moore. Formal Modelling of Large Domains — with an Application to Airline Business. Technical Report 74, UNU/IIST, P.O.Box 3058, Macau, June 1996. Revised: September 1996. .

[BE10] Dines Bjørner and Asger Eir. Compositionality: Ontology and Mereology of Domains. Some Clarifying Observations in the Context of Software Engineering in July 2008, eds. Martin Steffen, Dennis Dams and Ulrich Hannemann. In *Festschrift for Prof. Willem Paul de Roever Concurrency, Compositionality, and Correctness*, volume 5930 of *Lecture Notes in Computer Science*, pages 22–59, Heidelberg, July 2010. Springer.

[Bek84] Hans Bekič. Programming Languages and Their Definition. In Cliff B. Jones, editor, *Lecture Notes in Computer Science, Vol. 177*. Springer, 1984.

[BGH+97] Dines Bjørner, Chris W. George, Bo Stig Hansen, Hans Laustrup, and Søren Prehn. A Railway System, Coordination'97, Case Study Workshop Example. Research Report 93, UNU/IIST, P.O.Box 3058, Macau, January 1997. .

[BGP95] Dines Bjørner, Chris W. George, and Søren Prehn. Scheduling and Rescheduling of Trains. Research Report 52, UNU/IIST, P.O.Box 3058, Macau, December 1995. .

[Bjø77] Dines Bjørner. Programming Languages: Formal Development of Interpreters and Compilers. In *International Computing Symposium 77 (eds. E. Morlet and D. Ribbens)*, pages 1–21. European ACM, North-Holland Publ.Co., Amsterdam, 1977.

[Bjø97] Dines Bjørner. Michael Jackson's Problem Frames: Domains, Requirements and Design. In Li ShaoYang and Michael Hinchley, editors, *ICFEM'97: International Conference on Formal Engineering Methods*, Los Alamitos, November 12–14 1997. IEEE Computer Society. Final Version.

[Bjø03] Dines Bjørner. Domain Engineering: A "Radical Innovation" for Systems and Software Engineering ? In *Verification: Theory and Practice*, volume 2772 of *Lecture Notes in Computer Science*, Heidelberg, October 7–11 2003. Springer–Verlag. The Zohar Manna International Conference, Taormina, Sicily 29 June – 4 July 2003. .

[Bjø06] Dines Bjørner. *Software Engineering, Vol. 3: Domains, Requirements and Software Design*. Texts in Theoretical Computer Science, the EATCS Series. Springer, 2006.

[Bjø07] Dines Bjørner. Domain Theory: Practice and Theories, Discussion of Possible Research Topics. In *ICTAC'2007*, volume 4701 of *Lecture Notes in Computer Science (eds. J.C.P. Woodcock et al.)*, pages 1–17, Heidelberg, September 2007. Springer.

[Bjø08] Dines Bjørner. From Domains to Requirements. In *Montanari Festschrift*, volume 5065 of *Lecture Notes in Computer Science (eds. Pierpaolo Degano, Rocco De Nicola and José Meseguer)*, pages 1–30, Heidelberg, May 2008. Springer.

[Bjø09] Dines Bjørner. On Mereologies in Computing Science. In *Festschrift: Reflections on the Work of C.A.R. Hoare*, History of Computing (eds. Cliff B. Jones, A.W. Roscoe and Kenneth R. Wood), pages 47–70, London, UK, 2009. Springer.

[Bjø10a] Dines Bjørner. Domain Engineering. In Paul Boca and Jonathan Bowen, editors, *Formal Methods: State of the Art and New Directions*, Eds. Paul Boca and Jonathan Bowen, pages 1–42, London, UK, 2010. Springer.

[Bjø10b] Dines Bjørner. The Rôle of Domain Engineering in Software Development. Why Current Requirements Engineering Seems Flawed! In *Perspectives of Systems Informatics*, volume 5947 of *Lecture Notes in Computer Science*, pages 2–34, Heidelberg, Wednesday, January 27, 2010. Springer.

[Bjø11a] Dines Bjørner. Believable Software Management. *Encyclopedia of Software Engineering*, 1(1):1–32, 2011.

[Bjø11b] Dines Bjørner. Domains: Their Simulation, Monitoring and Control – A Divertimento of Ideas and Suggestions. In *Rainbow of Computer Science, Festschrift for Hermann Maurer on the Occasion of His 70th Anniversary.*, Festschrift (eds. C. Calude, G. Rozenberg and A. Saloma), pages 167–183. Springer, Heidelberg, Germany, January 2011.

[Bjø13] Dines Bjørner. *Domain Science and Engineering as a Foundation for Computation for Humanity*, chapter 7, pages 159–177. Computational Analysis, Synthesis, and Design of Dynamic Systems. CRC [Francis & Taylor], 2013. (eds.: Justyna Zander and Pieter J. Mosterman).

[Bjø14a] Dines Bjørner. *A Rôle for Mereology in Domain Science and Engineering*. Synthese Library (eds. Claudio Calosi and Pierluigi Graziani). Springer, Amsterdam, The Netherlands, October 2014.

[Bjø14b] Dines Bjørner. Domain Analysis: Endurants – An Analysis & Description Process Model. In Shusaku Iida, José Meseguer, and Kazuhiro Ogata, editors, *Specification, Algebra, and Software: A Festschrift Symposium in Honor of Kokichi Futatsugi*. Springer, May 2014.

[Bjø14c] Dines Bjørner. Domain Engineering – A Basis for Safety Critical Software. Invited Keynote, ASSC2014: Australian System Safety Conference, Melbourne, 26–28 May, December 2014.

[Bjø16a] Dines Bjørner. From Domain Descriptions to Requirements Prescriptions – A Different Approach to Requirements Engineering. *Submitted for consideration by Formal Aspects of Computing*, 35 pages. 2016.

[Bjø16b] Dines Bjørner. Manifest Domains: Analysis & Description. *Expected published by Formal Aspects of Computing*, 44 pages. 2016.

[BO80] Dines Bjørner and Ole N. Oest, editors. *Towards a Formal Description of Ada*, volume 98 of *LNCS*. Springer, 1980.

[CV99] R. Casati and A. Varzi. *Parts and Places: the structures of spatial representation*. MIT Press, 1999.

[DCT+96] Do Tien Dung, Le Linh Chi, Nguyen Le Thu, Phung Phuong Nam, Tran Mai Lien, and Chris W. George. Developing a Financial Information System. Technical Report 81, UNU/IIST, P.O.Box 3058, Macau, September 1996.

[DG96] Roderick Durmiendo and Chris W. George. Formal Development of a Digital Mutiplexed Radio-Telephone System. Research Report 67, UNU/IIST, P.O.Box 3058, Macau, Feb 1996. .

[JA97] Tomasz Janowski and Rumel V. Atienza. A Formal Model For Competing Enterprises, Applied to Marketing Decision-Making. Research Report 92, UNU/IIST, P.O.Box 3058, Macau, January 1997. .

[Lan64] Peter J. Landin. The Mechanical Evaluation of Expressions. *Computer Journal*, 6(4):308–320, 1964.

[Lan65a] Peter J. Landin. A Correspondence Between ALGOL 60 and Church's Lambda-Notation (in 2 parts). *Communications of the ACM*, 8(2-3):89–101 and 158–165, Feb.-March 1965.

[Lan65b] Peter J. Landin. A Generalization of Jumps and Labels. Technical report, Univac Sys. Prgr. Res. Grp., N.Y., 1965.

[LM97] Hoang Thi Tung Lam and Richard Moore. Specification of a Switching Communications System. Technical Report 106, UNU/IIST, P.O.Box 3058, Macau, May 1997.

[McC60] John McCarthy. Recursive Functions of Symbolic Expressions and Their Computation by Machines, Part I. *Communications of the ACM*, 3(4):184–195, 1960.

[McC62] John McCarthy. Towards a Mathematical Science of Computation. In C.M. Popplewell, editor, *IFIP World Congress Proceedings*, pages 21–28, 1962.

[MP66] John McCarthy and James Painter. Correctness of a Compiler for Arithmetic Expressions. In *[Sch67]*, pages 33–41, 1966. Dept. of Computer Science, Stanford University, California, USA.

[Rey70] John C. Reynolds. GEDANKEN – a simple type-less language based on the principle of completeness and the reference concept. *Communications of the ACM*, 13(5):308–319, 1970.

[Rey72] John C. Reynolds. Definitional Interpreters for Higher-Order Programming Languages. In *Proc. 25th ACM Nat'l. Conf.*, pages 717–740, 1972.

[Sch67] J.T. Schwartz. *Mathematical Aspects of Computer Science, Proc. of Symp. in Appl. Math.* American Mathematical Society, Rhode Island, USA, 1967.

[Sco70] D.S. Scott. Outline of a Mathematical Theory of Computation. In *Proc. 4th Ann. Princeton Conf. on Inf. Sci. and Sys.*, page 169, 1970.

[Sco72] D.S. Scott. Mathematical concepts in programming language semantics. In *Proc. AFIPS, Spring Joint Computer Conference, 40*, pages 225–234, 1972.

[SS71] D.S. Scott and C. Strachey. Towards a mathematical semantics for computer languages. In *Computers and Automata*, volume 21 of *Microwave Research Inst. Symposia*, pages 19–46, 1971.

[Str68] C. Strachey. Fundamental concepts in programming languages. Unpubl. Lecture Notes, NATO Summer School, Copenhagen, 1967, and Programming Research Group, Oxford Univ., 1968.

[VGJM02] Hung Dang Van, Chris George, Tomasz Janowski, and Richard Moore, editors. *Specification Case Studies in RAISE*. Springer, 2002.