

Dines Bjørner's Notes on SEMAT

Abstract

This note includes a position statement of the kind, I think, called for by Ivar Jacobsen in connection with the SEMAT initiative.

Contents

1	Position Statement	3
1.1	Background	3
1.2	Kernel	3
1.2.1	Software Development Techniques	3
1.2.2	Software Project Management	4
1.3	SEMAT Issue: Scope	6
A	For Your Information	7
B	Some Definitions	8
B.1	Method	8
B.2	Theory	9
B.3	Formal Specification Language	9
B.4	Engineering	10
B.5	Software Engineering	10
B.5.1	Definition of SE	10
B.5.2	An SE Dogma	11
B.5.3	The SE Triptych	11
B.6	SE Management	11
B.6.1	Software	12
C	A SEMAT Kernel	13
C.1	Semiotics	13
C.1.1	Syntax	13
C.1.2	Semantics	13
C.1.3	Pragmatics	13
C.2	Abstraction & Modelling	14
C.2.1	Abstraction	14
C.2.2	Models & Modelling	14
C.3	Software Engineering Process Models	15
C.3.1	Domain Engineering	15
C.3.2	Requirements Engineering	15
C.3.3	Software Design	16
C.4	Management	16
C.4.1	Software Project Management	16
C.4.2	Software Product Management	16

D Formal Specification Languages and Tools	17
D.1 Formal Specification Languages	17
D.2 Tools	17
E Bibliographical Notes	18
F CV: Dines Bjørner	21

1 Position Statement

1.1 Background

I shall focus on what Daniel Jackson

- Daniel Jackson [29]:
A direct Path to Dependable Software
CACM, Vol. 52, No. 4, pp 78-88, April 2009.

calls *Direct Paths* to software development and the *Direct Evidence* that software meets customer expectations and is correct with respect to requirements.

My position statement is, of course, heavily flavoured – read: biased – by almost 37 years of researching, teaching and directing software development projects that use formal techniques (to wit: VDM [8, 9, 21] and RAISE [23, 24]) leading to recent publications [2, 3, 4, 6]:

- Dines Bjørner. *Software Engineering, Vol. 1: Abstraction and Modelling*. Texts in Theoretical Computer Science, the EATCS Series. Springer, 2006.
- Dines Bjørner. *Software Engineering, Vol. 2: Specification of Systems and Languages*. Texts in Theoretical Computer Science, the EATCS Series. Springer, 2006. Chapters 12–14 are primarily authored by Christian Krog Madsen.
- Dines Bjørner. *Software Engineering, Vol. 3: Domains, Requirements and Software Design*. Texts in Theoretical Computer Science, the EATCS Series. Springer, 2006.
- Dines Bjørner. From Domains to Requirements — On a Triptych of Software Development. *Communications of the ACM*, 53(4), 2010. Submitted December 2009.

My position on software project management is very much influenced by Watts Humphrey:

- Dines Bjørner. Believable Software Management. *Encyclopedia of Software Engineering*, 1(1):1–32, 201. (Taylor & Francis, New York and London, edited by Philip Laplante).

1.2 Kernel

I shall focus on two aspects of what I consider a kernel: *the software development techniques* and *the software project management* aspects.¹

1.2.1 Software Development Techniques

It is my position – as also outlined in Appendix Sect. B (Pages 8–16) – that software development can and should be based on the use of formal techniques, whether “formal techniques” “lite””, i.e., systematically, rigorously or formally².

¹Section B.5 on page 10 deals with *software development techniques* and Sect. C.4 on page 16 deals – briefly – with *software project management*.

²Appendix Sect. B.4 on page 10, **Comment 9**, defines the terms: ‘systematically’, ‘rigorously’ and ‘formally’.

It is my position that many/most of the prevailing ‘methods’³ today can “smoothly” be made to fit the Triptych paradigm of software engineering (from domains via requirements to software design) outlined in Appendix Sects. B.5.1–B.5.3.

Comment 1

Many SEMAT “followers” may very well “shrink” away from my stating the above. So be it. But I see no other viable alternative when it comes to asking for software development to be pursued on a theoretical basis.

The formal techniques listed in Appendix Sect. D.1 on page 17 represent 40 years of academic research. Many are deployed in European software houses – to a smaller or larger extent [32]. Most of the ‘methods’ listed in footnote ?? can be rather smoothly subsumed by the formal techniques of Appendix Sect. D.1 on page 17⁴

End of Comment 1

I refer to the recently published collection [10].

1.2.2 Software Project Management

It is my position that the formal methods listed in Appendix Sect. D.1 on page 17 (with a few listed in Footnote 4) can all fit into the management principles of Watts Humphrey’s CMM, see my paper on this: [5].

Comment 2

Since, for example Watts Humphrey’s CMM concepts of process assessment and process improvements are commensurate with the formal methods listed in Appendix Sect. D.1 on page 17, I see no reason why many other management techniques cannot be so adapted.

End of Comment 2

Comment 3

No process model, no development approach can be scientifically proved to address the issues raised by Capers Jones in his 20 December 2010 (14:47 CET) e-mail:

1. Unstable, changing requirements = 95% of cases
2. Inadequate quality control and poor quality measures = 90%
3. Inadequate progress tracking = 85%
4. Inadequate cost and schedule estimating = 80%
5. False promises by outsource marketing and sales personnel = 80%
6. Rejecting good schedule estimates and replacing them with arbitrary dates = 75%

³– such as (alphabetically listed) agile, aspect-oriented, chaos model, evolutionary development, evolutionary prototyping, ICONIX Process (UML-based object modeling with use cases), incremental funding methodology, iterative processes, model driven development, prototyping, service-oriented modeling framework, software development rhythms, top-down and bottom-up design, Unified Process (UML), user experience, V-model, waterfall, XP (extreme programming), etcetera.

⁴Alloy, Event B, RAISE, VDM, Z etcetera.

7. *Informal, unstructured development = 70%*
8. *Inexperienced clients who can't articulate requirements = 60%*
9. *Inexperienced project managers = 50%*
10. *Inadequate tools for quality, static analysis, plus lack of inspections = 55%*
11. *Reusing materials filled with bugs = 30%*
12. *Inexperienced, unqualified software engineering teams = 20%*

But, one-by-one, one can argue, for each formal methods approach the extent to which it “solves” the issues raised by Capers Jones.

Similarly one can argue that the desirable (2025) software project characteristics listed (also) by Capers Jones in his position paper of 22 December 2010 can be achieved by these formal techniques:

1. *The cost of innovation and new features.*
2. *The cost of renovating legacy applications.*
3. *The cost of customer support after deployment.*
4. *The cost of creating and utilizing reusable components.*
5. *The cost of meetings and communications.*
6. *The cost of avoiding security flaws.*
7. *The cost of learning and training.*
8. *The cost of project management.*
9. *The cost of requirements changes.*
10. *The cost of producing English words.*
11. *The cost of programming or coding.*
12. *The cost of finding and fixing bugs.*
13. *The cost of security flaws and attacks.*
14. *The cost of cancelled projects.*
15. *The cost of litigation for failures and disasters.*

Thus I am in rather complete agreement with Watts Humphrey's (7 January 2010 20:23 CET) position statement:

1. **Goal:** *Everybody on the team knows the teams goal and what it takes to reach it.*
2. **Roles:** *All members know their personal roles on the team as well as the roles of all the other team members.*
3. **Strategy:** *All team members know and agree with the overall team strategy and their role in supporting it.*
4. **Process:** *Everybody knows how to do their own job and how everybody else does their jobs.*
5. **Plan:** *Everybody knows what to do at all times and nobody stands around waiting to be told their next assignment.*
6. **Support:** *Everybody is aware of team workload and is prepared to pitch in and help whenever somebody needs a hand.*
7. **Status:** *Everybody knows precisely where the team stands at all times and is prepared to make an extra effort whenever needed to achieve overall team success.*

As well as his (differently numbered):

- **Preparation Tasks:**

1. Define the projects goals. What is it that the team is to do?
2. Define the team, its members, its roles, and its scope. What development functions are represented on the team such as testing, software development, hardware development, or systems engineering, and what responsibilities will the team and its members have?
3. Establish the development strategy. How does the team intend to do the job, are prototypes needed, how many releases are required, what cycles are planned, and what is cycle scope and duration?
4. Produce a list the products to be produced and their essential characteristics, like size, function, and principal specifications.
5. Produce the team plan. What are the tasks to be performed for each process step and what effort will be required for each task and product element?
6. Obtain management agreement to the team plan. Does management agree with the teams plan, are revisions needed, and does the team agree with the revisions?

- **Development and Development Management Tasks:**

9. The team performs the development work.
10. As the work proceeds, the team adjusts the plan and work assignments to conform to project status and the team members current understanding of the work.
11. The team regularly reports its progress to management.
12. The team monitors risks and issues and obtains management assistance in resolving problems that it cannot handle.
13. The team dynamically replans the work as requirements, team membership, product knowledge, and development status change.

- **Assessment Activities:**

14. Following completion of each major project milestone, the team analyzes its performance, identifies areas for improvement, gathers data on project results, and documents lessons learned.

End of Comment 3

1.3 SEMAT Issue: Scope

I cover a few of the SEMAT issues, such as outlined in Ivar, Bertrand and Richard's 12 Januar 2010 (22:11 CET) e-mail.

My position is that some such *direct path* to software as that of, for example, the **Triptych** approach with its attendant management issues sets a suitable scope. The 'univerdals' called for in Ivar, Bertrand and Richard's 12 Januar 2010 (22:11 CET) e-mail, I maintain, are those of *domain engineering*, *requirements engineering*, *software design*, *phases*, *stages*, *steps*, etc., etc. as outlined in Sect. B, as well as the concepts of *direct path* and *direct evidence* as outlined in Daniel Jackson's 2009 CACM article [29].

End of Dines Bjørner's Position Statement

A For Your Information

Since I may very well be only vaguely known to most of the SEMAT initiative supporter I provide, in Sect. F on page 21, a CV.

Let me emphasize that through my work, during the 1980s, at Dansk Datamatik Center (notably its Ada compiler development projects), and through my leadership, during the 1990s, founding an first UN Director, of UNU-IIST (www.iist.unu.edu), I have quite some real experience with all issues of software engineering. As an educator I have been able to challenge my MSc and PhD students in such ways that more than 100 of my MSc and PhD theses candidates are now working for a number of software houses that my students have founded (DDCI Inc., Maconomy, PDC (Prolog Development Center), C-Brain, etc., and core departments of major software houses in Denmark (Terma Space Division, etc.).

I have with the DDC Ada, CHILL and RAISE projects, and with development projects for the Philippine PTT, Vietnam ministry of finance, Chinese ministry of railways, etc., instigated, led and concluded very successful software development projects: on time, at cost, basically correct and fully meeting customers' expectations. These projects have all been based on the view and approach of software engineering, including its use of formal techniques, outlined in this note.

My major references are:

1. Dines Bjørner. *Software Engineering, Vol. 1: Abstraction and Modelling*. Texts in Theoretical Computer Science, the EATCS Series. Springer, 2006.
2. Dines Bjørner. *Software Engineering, Vol. 2: Specification of Systems and Languages*. Texts in Theoretical Computer Science, the EATCS Series. Springer, 2006. Chapters 12–14 are primarily authored by Christian Krog Madsen.
3. Dines Bjørner. *Software Engineering, Vol. 3: Domains, Requirements and Software Design*. Texts in Theoretical Computer Science, the EATCS Series. Springer, 2006.
4. Dines Bjørner. Believable Software Management. *Encyclopedia of Software Engineering*, 1(1):1–32, 201. (Taylor & Francis, New York and London, edited by Philip Laplante).
5. Dines Bjørner. From Domains to Requirements — On a Triptych of Software Development. *Communications of the ACM*, 53(4), 2010. Submitted December 2009.

I attach documents 4. and 5. when e-mailing the present note.

Document 5. is attached in two versions: a short, 8 page, for CACM (hopefully) and a long, 11.5 page, with supporting formulas. The short, when contrasted to the long, shows that one can indeed get quite far with serious software engineering without necessarily formalising — but, as we all ought know by now, one cannot get as far as with formalisations.

B Some Definitions

Three terms appears crucial to the SEMAT initiative: SE: software engineering, M: method, and T: theory. We shall define these terms carefully, in the order: M, T and SE. Auxiliary terms are also defined in this section.

B.1 Method

By a method we shall understand:

- a set of principles
- for selecting and applying
- a number of techniques
- for analysing a problem and
- synthesizing
- an artifact (here software).

I refer to a recent paper:

- Daniel Jackson [29]:
A direct Path to Dependable Software
CACM, Vol. 52, No. 4, pp 78-88, April 2009.

I am of the strong conviction that some form of adoption by the SEMAT initiative, of the ideas brought forward in this paper, is crucial.

Comment 4

As already and abundantly noted by many SEMAT discussants, there are a great variety of 'software engineering cum software development methods' "around". Many of these methods ($M_i = \{m_{i_1}, m_{i_2}, \dots, m_{i_{n_i}}\}$) are based on (no doubt) sound and valid observations by industry practitioners. And some of these methods ($M_a = \{m_{a_1}, m_{a_2}, \dots, m_{a_{n_a}}\}$) are based on (no doubt) sound and valid theoretical research by university academics. It seems that neither $M_i \not\subseteq M_a$ nor $M_a \not\subseteq M_i$ is the case, and that some would say that $M_i \cap M_a = \{\}$. I reckon that the Agile, Aspect, ..., UML methods belong to M_i , and I reckon that the Event B, RAISE, VDM, Z methods belong to M_a . It is clear to me that the Event B, RAISE, VDM, Z methods all have a theoretical foundations. It is not clear to me whether any or all of the Agile, Aspect, ..., UML possess such a foundation⁵.

End of Comment 4

⁵By using ellipses in the listing Agile, Aspect, ..., UML I can, of course, always claim this "lack" of foundation!

B.2 Theory

By a theory shall understand:

- a proof system consisting
 - of axioms
 - and deduction rules

and

- a set of theorems (developed using the proof system).

Usually the proof system is related to a specific formal specification language.

The theorems are therefore usually statements about of what properties specifications are aiming at specifying.

Comment 5

When the SEMAT initiative calls for “a theory” I assume that it means “a theoretical foundation”. An engineering approach, an engineering method, when based on a theoretical foundation, need not imply that the practitioners of this method need understand the usually mathematical foundations of such a theory. Aeronautics engineering is based on aerodynamics. The aeronautics engineer need only once have understood the natural science of aerodynamics. In her professional life the aeronautics engineer uses tools and techniques that build on the natural science of aerodynamics. These aeronautics engineer tools and techniques themselves require that the aeronautics engineer knows the related mathematics (Navier Stokes [differential] equations, etcetera). For the software engineer I envisage the kind of tools and techniques that will be mentioned in this note: specification languages and proof systems, respectively abstraction, modelling, formal testing, model checking, verification, etcetera.

End of Comment 5

A theory must necessarily fulfill (a number of) the following criteria:

- Logically consistent
- Consistent with accepted facts
- Testable
- Parsimonious
- Consistent with related theories
- Interpretable: explain and predict
- Pleasing to the mind (Esthetic, Beautiful)
- Useful (Applicable)

B.3 Formal Specification Language

Comment 6

We take it as an accepted fact that a software engineering method that needs a theory (i.e., satisfies the SEMAT paradigm, includes the use of formal specification languages, even, and usually, in a “lite”-weight fashion: not asking the software engineer to “do math” all day long !

End of Comment 6

By a formal specification language we shall understand

- a formal,
- i.e., mathematical

definition of

- a syntax of that language,
- a semantics for that language, and
- a proof system for that language.

Comment 7

Section D lists a number of formal specification languages (Sect. D.1 on page 17) and tools (Sect. D.2 on page 17).

End of Comment 7

B.4 Engineering

- The engineer “walks the bridge”
- from science to technology
- in order to construct artifacts based on science,
- from technology to science
- in order to assess possible scientific properties of artifacts.

Comment 8

In this SEMAT note I advocate some form of use of formal techniques:

- *not necessarily **fully formal**, that is, there is no need to formally state all correctness theorems to be proven, let alone prove them, but **systematically**, i.e., “formal methods lite”, where the most relevant domain, the requirements and the software is formally specified, with hints of their relation – but probably just that;*

- *or, when greater care is called for, **rigorously**, with crucial, core parts of the domain, requirements and software design being subject to some analysis (model checking and verification proofs); or*
- **fully formal** – *where all phases, stages and steps are formalised, verified, etc.*

Usually a systematic, “light-weight” approach using formal techniques suffices.

End of Comment 8

B.5 Software Engineering

B.5.1 Definition of SE

By software engineering I understand

- the engineering, that is the sciences and pragmatics
 - applied in order to effectively construct effective software
 - that is the right software, i.e., meets customer expectations and only those,
 - and which leads to software that is correct with respect to customer requirements.

B.5.2 An SE Dogma

On the scientific side I, myself, is guided by the following approach to ensure that it is the right software and that software is right:

- before software can be designed
- we must ensure that we have a robust understanding of its requirements,
- and to prescribe the right requirements
- we must ensure that we have a robust understanding of the underlying domain.

B.5.3 The SE Triptych

I therefore advocate [2, 3, 4, 6] a set of

- phases,
- stages and
- steps

of development basically “sequenced” through the phases of

- domain engineering,
- requirements engineering and
- software design.

Each phase, stage and step:

- results in documents,
- these documents are
 - both informal, clear, say English texts (narratives)
 - and accompanying formal specifications
 - with test, model-checking and formal verification documents,
 - etcetera.

B.6 SE Management

- SE management is about
 - the management of resources, their
 - * planning,
 - * scheduling,
 - * allocation,
 - * monitoring and
 - * control;
 - the management of the logics of the developed artifacts:
 - * domain, requirements and software design documents and
 - * QA, quality assurance, etc.
- I refer to the article: [5]
 - Dines Bjørner
Believable Software Management
Encyclopedia of Software Engineering
Taylor & Francis, New York and London
edited by Philip Laplante, 2010

This article is based on Watts Humphrey's concept of *Capability Maturity Management*.

- The article show that pragmatic, sound management concepts
- can be applied rigorously
- to formal techniques-based software developments
- such as those advocated by this SEMAT note.

B.6.1 Software

By software I shall understand

- not only
 - the executable code (i.e., image)
 - and its (possibly electronic) installation and user manuals
- but also all the documents that arose as a result of the development of this software:
 - domain (D), requirements (R) and software (S) design specifications,

- * stake-holder docs.,
 - * acquisition docs.,
 - * analysis docs.,
 - * terminology docs.,
 - * descriptions,
 - * prescriptions,
 - * design, etc.
- all R, R and S related tests
(test cases and test outcomes, whether successful or not),
 - all D, R and S related model checks, and
 - all D, R and S related theorems and their proofs;
- and all relevant manuals:
 - documents concerning portability and installation,
 - maintenance manuals (adaptive, corrective, perfective and preventive,
 - D, R and S development logbooks,
 - quality assurance, process assessment and process improvement reports,
 - and all related planning, allocation & scheduling as well as management documents.

C A SEMAT Kernel

By a ‘kernel’ for “SE methods needs theory” I shall understand a set of professional qualifications and the tools to support the professional deployment of these qualifications.

C.1 Semiotics

The professional software engineer, must, I advocate know what is meant by semiotics: the confluence of syntax, semantics and pragmatics.

C.1.1 Syntax

Definition I

By syntax we shall, in the context of textual or diagrammatic (spoken, written, programming, specification) languages understand the way in which linguistic elements (as words or as boxes and arrows) are put together to form constituents (as phrases or clauses or sub-diagrams). When dealing with systems such as the phenomenological structures of mechanical machinery, architectural buildings, or transportation networks, or such as conceptual structures of the financial service industry, the health care industry, transportation logistics we shall by syntax mean the way in which parts of these structures relate to one-another and to the whole (aka. mereology).

The professional software engineer, must, I advocate know principles, techniques and tools of defining and using various forms of formal syntax, incl. BNF, abstract syntax and XML.

C.1.2 Semantics

Definition II

By semantics we shall understand the relations between syntactic signs (words, terms) and what they refer to and including theories of denotation, extension, naming, and truth.

The professional software engineer, must, I advocate know principles, techniques and tools of defining and using various forms of formal semantics: denotational and operational (say transition system and SECD machine) semantics.

C.1.3 Pragmatics

Definition III

By pragmatics we shall understand the relation between signs or linguistic (or diagrammatic) expressions and their users. Pragmatics is concerned with the relationship of sentences to the environment in which they occur.

The professional software engineer, must, I advocate know principles of pragmatics in order not to transgress the meta-linguistically different levels between syntax and semantics, at one level, and pragmatics at another level, that is, in order not to bring elements of confusion into domain descriptions and requirements prescriptions.

Comment 9

We use words or diagrams in order to communicate or designate. We mean the semantics of what has been communicated or designated. And pragmatics is why we uttered the words in the first place.

End of Comment 9

C.2 Abstraction & Modelling

The professional software engineer, must, I advocate know of well-documented principles and techniques for abstraction and modelling – as expressible both in natural language, i.e., English and in a suitable repertoire of formal specification languages, textual and diagrammatic.

C.2.1 Abstraction**Comment 10**

Abstraction is a tool, used by the human mind, and to be applied in the process of describing (understanding) complex phenomena. Abstraction is the most powerful such tool available to the human intellect. Science proceeds by simplifying reality. The first step in simplification is abstraction. Abstraction (in the context of science) means leaving out of account all those empirical data which do not fit the particular, conceptual framework within which science at the moment happens to be working. Abstraction (in the process of specification) arises from a conscious decision to advocate certain desired objects, situations and processes as being fundamental; by exposing, in a first, or higher, level of description, their similarities and — at that level — ignoring possible differences.

End of Comment 10

C.2.2 Models & Modelling**Comment 11**

Models can be iconical, analogical, or analytical models. They can be descriptive, or prescriptive models, and extensional or intensional models. Specifications are what we write down, syntactically. Models are what these specifications denote, i.e., their meaning.

End of Comment 11

Models span the spectrum between

1. property-oriented and
2. model-oriented

specifications.

The former emphasise logical properties; the latter mathematical structures. The former could be said to be more proof-friendly; the latter to be more implementation-friendly. There are many techniques to refining the former kind of specifications to the latter kind.

C.3 Software Engineering Process Models

Comment 12

We take it as a fundamental dogma – to be followed by the professional software engineer – that software is developed in phases, stages and steps. In “ye olde days” one referred to process models⁶. Common to all these is the notion of stages (phases or steps) of development, from more abstract to more concrete. We shall try summarise “all” of these approaches by listing the phases, stages and steps of a Triptych approach to SE, one in terms of which many of the major ideas of each of the footnoted (Footnote 6) process models can be understood. The “theoretical” Triptych model – in its practical adaptations “smoothly” allow for several of the main ideas of footnoted process models.

End of Comment 12

The professional software engineer, must, I advocate know the major principles, techniques and tools of the enumerated items listed below.

C.3.1 Domain Engineering

1. Stake-holder liaison
2. Domain Acquisition and Analysis
3. Business Processes
4. Domain Modelling: Constructing Domain Descriptions
5. Model Analysis
 - (a) Testing
 - (b) Model-checking
 - (c) Verification
 - (d) Validation

C.3.2 Requirements Engineering

1. Stake-holder liaison
2. Requirements Acquisition and Analysis
3. Business Process Re-engineering
4. Requirements Modelling: Constructing Requirements Prescriptions
5. Model Analysis

⁶– such as (alphabetically listed) agile, aspect-oriented, chaos model, evolutionary development, evolutionary prototyping, ICONIX Process (UML-based object modeling with use cases), incremental funding methodology, iterative processes, model driven development, prototyping, service-oriented modeling framework, software development rhythms, top-down and bottom-up design, Unified Process (UML), user experience, V-model, waterfall, XP (extreme programming), etcetera.

- (a) Testing
- (b) Model-checking
- (c) Verification
- (d) Validation

C.3.3 Software Design

Et cetera.

More to come.

C.4 Management

C.4.1 Software Project Management

Reference is made to [5, Bjørner: *Believable Software Management*].

More to come.

C.4.2 Software Product Management

More to come.

D Formal Specification Languages and Tools

D.1 Formal Specification Languages

- Alloy [30]
- B, Event B [1]
- CSP [27]
- DC (Duration Calculus) [42]
- MSC (Message Sequence Charts) [28]
- Petri Nets [37]
- RAISE, RSL [23, 24, 2, 3, 4]
- Statecharts [26]
- TLA+ (Temporal Logic of Actions “+”) [31]
- VDM, VDM-SL [8, 9, 21]
- Z [41]

D.2 Tools

Besides the usual tools “surrounding” most formal specification languages, to wit:

- | | |
|--------------------|---------------|
| • Alloy [30] | • TLA+ [31] |
| • B, Event B [1] | • VDM-SL [21] |
| • FDR/CSP [38, 22] | • Z [41] |
| • RSL [24] | |

there are a number of language-independent tools for model-checking and proofs:

- NuSMV (New SMV)
<http://en.wikipedia.org/wiki/NuSMV>
- PVS (Prototype Verification System)
[34, 35, 36, 39, 40]
- The SPIN Model-checking Verification System) [25]
- STeP (Stanford Temporal Logic Prover &c.) [33]

E Bibliographical Notes

References

- [1] Jean-Raymond Abrial. *The B Book: Assigning Programs to Meanings and Modeling in Event-B: System and Software Engineering*. Cambridge University Press, Cambridge, England, 1996 and 2009.
- [2] Dines Bjørner. *Software Engineering, Vol. 1: Abstraction and Modelling*. Texts in Theoretical Computer Science, the EATCS Series. Springer, 2006.
- [3] Dines Bjørner. *Software Engineering, Vol. 2: Specification of Systems and Languages*. Texts in Theoretical Computer Science, the EATCS Series. Springer, 2006. Chapters 12–14 are primarily authored by Christian Krog Madsen.
- [4] Dines Bjørner. *Software Engineering, Vol. 3: Domains, Requirements and Software Design*. Texts in Theoretical Computer Science, the EATCS Series. Springer, 2006.
- [5] Dines Bjørner. Believable Software Management. *Encyclopedia of Software Engineering*, 1(1):1–32, 201. (Taylor & Francis, New York and London, edited by Philip Laplante).
- [6] Dines Bjørner. From Domains to Requirements — On a Triptych of Software Development. *Communications of the ACM*, 53(4), 2010. Submitted December 2009.
- [7] Dines Bjørner. *Domain Engineering: Technology Management, Research and Engineering*. JAIST Press, March 2009. The monograph contains the following chapters: [12, 13, 14, 15, 11, 16, 17, 18, 19, 20].
- [8] Dines Bjørner and Cliff B. Jones, editors. *The Vienna Development Method: The Meta-Language*, volume 61 of *LNCS*. Springer, 1978.
- [9] Dines Bjørner and Cliff B. Jones, editors. *Formal Specification and Software Development*. Prentice-Hall, 1982.
- [10] Paul Boca, Jonathan P. Bowen, and Jawed I. Siddiqi, editors. *Formal Methods: State of the Art and New Directions*. Springer, London, UK, 2010. ISBN 978-1-84882-735-6.
- [11] Dines Bjørner. *Domain Engineering: Technology Management, Research and Engineering*, chapter 5: The Triptych Process Model – Process Assessment and Improvement, pages 107–138. JAIST Press, March 2009.
- [12] Dines Bjørner. *Domain Engineering: Technology Management, Research and Engineering [7]*, chapter 1: On Domains and On Domain Engineering – Prerequisites for Trustworthy Software – A Necessity for Believable Management, pages 3–38. JAIST Press, March 2009.
- [13] Dines Bjørner. *Domain Engineering: Technology Management, Research and Engineering [7]*, chapter 2: Possible Collaborative Domain Projects – A Management Brief, pages 39–56. JAIST Press, March 2009.
- [14] Dines Bjørner. *Domain Engineering: Technology Management, Research and Engineering [7]*, chapter 3: The Rôle of Domain Engineering in Software Development, pages 57–72. JAIST Press, March 2009.

- [15] Dines Bjørner. *Domain Engineering: Technology Management, Research and Engineering [7]*, chapter 4: Verified Software for Ubiquitous Computing – A VSTTE Ubiquitous Computing Project Proposal, pages 73–106. JAIST Press, March 2009.
- [16] Dines Bjørner. *Domain Engineering: Technology Management, Research and Engineering [7]*, chapter 6: Domains and Problem Frames – The Triptych Dogma and M.A.Jackson’s PF Paradigm, pages 139–175. JAIST Press, March 2009.
- [17] Dines Bjørner. *Domain Engineering: Technology Management, Research and Engineering [7]*, chapter 7: Documents – A Rough Sketch Domain Analysis, pages 179–200. JAIST Press, March 2009.
- [18] Dines Bjørner. *Domain Engineering: Technology Management, Research and Engineering [7]*, chapter 8: Public Government – A Rough Sketch Domain Analysis, pages 201–222. JAIST Press, March 2009.
- [19] Dines Bjørner. *Domain Engineering: Technology Management, Research and Engineering [7]*, chapter 9: Towards a Model of IT Security — – The ISO Information Security Code of Practice – An Incomplete Rough Sketch Analysis, pages 223–282. JAIST Press, March 2009.
- [20] Dines Bjørner. *Domain Engineering: Technology Management, Research and Engineering [7]*, chapter 10: Towards a Family of Script Languages – – Licenses and Contracts – Incomplete Sketch, pages 283–328. JAIST Press, March 2009.
- [21] John Fitzgerald and Peter Gorm Larsen. *Modelling Systems – Practical Tools and Techniques in Software Development*. Cambridge University Press, Cambridge, UK, Second edition, 2009.
- [22] Formal Systems Europe. Home of the FDR2. Published on the Internet: <http://www.fsel.com/>, 2003.
- [23] Chris W. George, Peter Haff, Klaus Havelund, Anne Elisabeth Haxthausen, Robert Milne, Claus Bendix Nielsen, Søren Prehn, and Kim Ritter Wagner. *The RAISE Specification Language*. The BCS Practitioner Series. Prentice-Hall, Hemel Hempstead, England, 1992.
- [24] Chris W. George, Anne Elisabeth Haxthausen, Steven Hughes, Robert Milne, Søren Prehn, and Jan Storbank Pedersen. *The RAISE Method*. The BCS Practitioner Series. Prentice-Hall, Hemel Hempstead, England, 1995.
- [25] J.-C. Grégoire, G. J. Holzmann, and D. Peled, editors. *The SPIN Verification System*, volume 32 of *DIMACS series*. American Mathematical Society, 1997. ISBN 0-8218-0680-7, 203p.
- [26] David Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8(3):231–274, 1987.
- [27] Tony Hoare. *Communicating Sequential Processes*. C.A.R. Hoare Series in Computer Science. Prentice-Hall International, 1985. Published electronically: <http://www.usingcsp.com/csp-book.pdf> (2004).
- [28] ITU-T. CCITT Recommendation Z.120: Message Sequence Chart (MSC), 1992, 1996, 1999.

- [29] Daniel Jackson. A Direct Path to Dependable Software. *CACM: Communications of the ACM*, 52(4):78–88, April 2009.
- [30] Daniel Jackson. *Software Abstractions: Logic, Language, and Analysis*. The MIT Press, Cambridge, Mass., USA, April 2006. ISBN 0-262-10114-9.
- [31] Leslie Lamport. *Specifying Systems*. Addison–Wesley, Boston, Mass., USA, 2002.
- [32] Peter Gorm Larsen, John Fitzgerald, and Tom Brookes. Applying Formal Specification in Industry. *IEEE Software*, 13(3):48–56, May 1996.
- [33] Zohar Manna and Amir Pnueli. *The Temporal Logic of Reactive Systems: (vol.1: Specifications, vol.2: Safety)*. Addison Wesley, 1991 and 1995.
- [34] S. Owre, J. Rushby, and N. Shankar. PVS: Prototype Verification System. In *11th Intl. Conf. on Automated Deduction (CADE-11)*, LNCS 607; Lecture Notes in Computer Science, pages 748–752, Saratoga, NY., USA, 1995. Springer-Verlag.
- [35] S. Owre, N. Shankar, J. M. Rushby, and D. W. J. Stringer-Calvert. *PVS Language Reference*. Computer Science Laboratory, SRI International, Menlo Park, CA, September 1999.
- [36] S. Owre, N. Shankar, J. M. Rushby, and D. W. J. Stringer-Calvert. *PVS System Guide*. Computer Science Laboratory, SRI International, Menlo Park, CA, September 1999.
- [37] Wolfgang Reisig. *Petrinetze: Modellierungstechnik, Analysemethoden, Fallstudien*. Institut für Informatik, Humboldt Universität zu Berlin, Unter den Linden 6, 10099 Berlin, Germany, 1 Oktober 2009. 276 pages. http://www2.informatik.hu-berlin.de/top/pnene_buch/pnene_buch.pdf.
- [38] A. W. Roscoe. *Model checking CSP*, pages 353–378. Prentice-Hall Intl., 1994.
- [39] N. Shankar, S. Owre, and J. M. Rushby. *PVS Tutorial*. Computer Science Laboratory, SRI International, Menlo Park, CA, February 1993. Also appears in Tutorial Notes, *Formal Methods Europe '93: Industrial-Strength Formal Methods*, pages 357–406, Odense, Denmark, April 1993.
- [40] N. Shankar, S. Owre, J. M. Rushby, and D. W. J. Stringer-Calvert. *PVS Prover Guide*. Computer Science Laboratory, SRI International, Menlo Park, CA, September 1999.
- [41] J. C. P. Woodcock and J. Davies. *Using Z: Specification, Proof and Refinement*. Prentice Hall International Series in Computer Science, 1996.
- [42] Chao Chen Zhou and Michael R. Hansen. *Duration Calculus: A Formal Approach to Real-time Systems*. Monographs in Theoretical Computer Science. An EATCS Series. Springer–Verlag, 2004.

F CV: Dines Bjørner

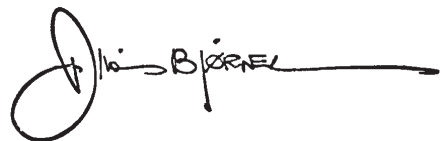
- **Family &c.:** Dines Bjørner (DB) was born in Odense, Denmark, 4 October 1937. His father had an MSc degree in Mathematics (from Copenhagen University, 1931) and his mother a BA degree in Nordic and Modern English/America Literature (also from Copenhagen University, 1929). Since 1965 DB has been married to Kari Skallerud Bjørner (Oslo, Norway). They have two children, Charlotte and Nikolaj, and five grandchildren.
- **Educational Background:** DB graduated, in 1956, with a senior high school degree in Mathematics and Natural Sciences from the Århus Cathedral School (founded in 1142). DB graduated in January 1962 with an MSc in Electronics Engineering and with a Ph.D. in Computer Science in January 1969 from the Technical University of Denmark (founded by Hans Christian Ørsted in 1828).
- **IBM Career:** DB joined IBM in March 1962 at their Nordic Laboratories (founded by Cai Kinberg) in Stockholm, Sweden (where DB also first met Jean Paul Jacob and Gunnar Wedell). DB was transferred to the IBM Systems Development Division (IBM SDD) at San Jose, California, USA, in December 1963. While doing his Ph.D. (September 1965–January 1969) DB was a lecturing consultant to IBM's European Systems Research Institute (ESRI) at Geneva, Switzerland (where DB received valuable guidance from Carlo Santacrose and where DB's friendship with Gerald Weinberg started) (1967–1968). In 1969 DB worked at IBM's Advanced Computing Systems (IBM ACS) Laboratory, Menlo Park, California, and, later that year until early 1973 at IBM Research, San Jose (again Jean Paul Jacob became a colleague). Transferred to the IBM Vienna Laboratory (directed then by Heinz Zemanek), Austria, DB resigned from IBM in August 1975 to return to Denmark after basically 13 years abroad.
- **Career Outside and After IBM:** During his stay at IBM Research DB was a visiting lecturer, for several quarters, at University of California at Berkeley (1971–1972), instigated by Lotfi Zadeh whom DB considers his main mentor and for whom DB has the fondest regards. DB was a visiting guest professor at Copenhagen University in the academic year 1975–1976, before taking up his chair in September 1976 at the Technical University of Denmark (DTU). During the summer semester of 1980 DB was the Danish Chair Professor at the Christian-Albrechts University of Kiel, Germany — hosted by Prof. Dr. Hans Langmaack. Together with a colleague, Prof. Christian Gram, DB instigated the Dansk Datamatik Center (DDC) in the summer of 1979. During the 1980s DB was chief scientist of DDC. In 1982–1984 DB was chairman of a Danish Government (Ministry of Education) Commission on Informatics. DB was the founding and first UN Director of UNU-IIST, the United Nations University's International Institute for Software Technology, located in Macau. DB was a visiting professor at NUS: National University of Singapore in the academic year 2004–2005, and a research guest professor at JAIST, Japan Advanced Institute of Science and Technology, Ishikawa Prefecture, Japan for basically the calendar year 2006 — where the work reported in this monograph was begun. DB was a visiting professor at Université Henri Poincaré and at INRIA/LORIA, Nancy, France, for two months: Oct.–Dec., 2007. During the fall and spring of 2008–2009 DB was lecturing at the Techn. Univ. of Graz, Austria and at University of Saarland, Saarbrücken, Germany (March 2009).
- **Lectures and Graduates:** DB has lectured and regularly lectures on six continents in almost

50 countries and territories and has advised more than 130 MSc's and almost two dozen PhDs.

- **Research &c. Work:** At IBM DB first worked in the hardware (logic and systems) design of such equipment as the IBM 1070 (Sweden), the IBM 1800 and IBM 1130 computers (San Jose), and, finally, with Gene Amdahl and Ed Sussenguth, on the IBM ACS/1 supercomputer (Menlo Park). At Research DB worked with the late John W. Backus and the late Ted Codd on Functional Languages, resp. Relational Data Base Systems. At Vienna, DB, together with such colleagues as Peter Lucas, the late Hans Bekič, Kurt Walk, and Cliff B. Jones, worked on a Denotational (–like) Semantics Description of PL/I while, with his colleagues conceiving, researching, developing and using VDM (the Vienna software Development Method). At DTU and at DDC, supported by the European Community, DB initiated several advanced research & development projects: (1) Formal Semantics Description of and (2) full language compiler for CHILL (the Intl. Telecommunications Unions Communications [C.C.I.T.T.] High Level Language) — both significantly developed by Peter L. Haff (and the late Søren Prehn); (3) Formal Semantics Description of and (4) the first European US DoD officially validated compiler for the US DoD Ada embedded systems programming language — with significant and indispensable contributions by DB's colleague Dr. Hans Bruun and, again, the late Søren Prehn; (5) RAISE (Rigorous Approach to Industrial Software Engineering, headed by the late Søren Prehn and Chris George); (6) Formal Semantics Definition of VDM–SL (the VDM Specification Language, Bo Stig Hansen and Peter Gorm Larsen); (7) ProCoS (Provably Correct Systems) with, amongst others, Profs. Sir Tony Hoare (then Oxford, now Microsoft Research, Cambridge, UK), Hans Langmaack (Kiel) and Ernst-Rüdiger Olderog (Oldenburg) and others.
- **UNU-IIST:** At UNU-IIST DB had a rather free hand, and was able, with a small team of excellent colleagues (Prof. Zhou Chaochen (Academician, the Chinese Academy of Science), the late Søren Prehn, Chris W. George, Richard Moore, Tomasz Janowski, Dang Van Hung, Xu Qi Wen and Kees Middelburg), to further explore the research issues still occupying DB's interest, and to apply them (i.e., test them out) in a number of joint R&D projects with institutions in developing and newly industrialised countries [including newly independent states] (Argentina, Belarus, Brasil, Cameroun, China, Gabon, India, Indonesia, Mongolia, North Korea, Pakistan, Philippines, Poland, Romania, Russia, South Africa, South Korea, Thailand, Vietnam, Ukraine, Uruguay, etc.).
- **Societal Work:** DB was a co-founder of VDM-Europe in 1987 and moved VDM-Europe onto FME: Formal Methods Europe in 1991. DB co-chaired two of the VDM Symposia (1987, 1990), and the International Conference on Software Engineering (ICSE) in 1989 in Pittsburgh, Pennsylvania, USA. DB was chairman of the IFIP World Congress in Dublin, Ireland in 1986, and was the instigator and General Chairman of the first World Congress on Formal Methods, FM'99, in Toulouse, France, September 20–24, 1999. DB has otherwise been involved in about 60 other scientific conferences.
- **Awards &c.:** DB is a Knight of The Danish Flag; is a member of Academia Europaea (MAE) and was chairman of its Informatics Section (2004–2009); is a member of The Russian Academy of Natural Sciences (MRANS [AB]), and of IFIP Working Groups 2.2 (1980–2004) and 2.3 (1980–2008). DB has received the John von Neumann Medal of the JvN Society of Hungary and the Ths. Masaryk Gold Medal from the Masaryk University,

Brno, The Czech Republic. DB received the Danish Engineering Society's (IDA) Informatics Division's (IDA-IT) first BIT prize, March 1999. DB was given the degree of honorary doctor from the Masaryk University, Brno, The Czech Republic, in 2004. DB is an ACM Fellow and an IEEE Fellow.

- **Publications:** DB has authored more than 120 published papers and co-authored and co-edited some 15 books and written three books [2, 3, 4, 7].
- **Research Interests:** DB's research interests, since his Vienna days, have centered on programming methodology: *Methods as sets of principles for selecting and applying mathematics-based analysis and construction techniques and tools in order efficiently to construct efficient artefacts* — notably software. DB sees his main contributions to be in the research, development and propagation of formal specification principles and techniques. Currently DB focuses on the triptych of domain engineering, requirements engineering and software architecture and program organisation methods — emphasising such that relate these in mathematical as well as technical ways: (1) Intrinsic, support technology, management & organisation, rules & regulation, and human behaviour facets of domains; (2) projection, instantiation, extension and initialisation of domain requirements, etc.; (3) software architectures as refinements of domain requirements, and program organisation as refinements of machine requirements — with interface requirements (currently) being refinements of either and both!
- **Acknowledgements:** Among the very many people for whom DB has a special, professional fondness, people who have helped DB in his professional career, he wishes to bear tribute, in approximate chronological order, to (the late) Cai Kinberg, Gunnar Wedell, Jean Paul Jacob, Gerald Weinberg, Gene Amdahl, Ed Sussenguth, Tien Chi (T.C.) Chen, Lotfi Zadeh, (the late) Ted Codd, (the late) John W. Backus, Peter Lucas, Cliff Jones, (the late) Hans Bekič, Kurt Walk, Christian Gram, Ole N. Oest, Erich Neuhold, (the late) Søren Prehn, Sir Tony Hoare, Hans Langmaack, Zhou Chao Chen, Chris George and Kokichi Futatsugi.

A handwritten signature in black ink, appearing to read 'John B. Journer'. The signature is stylized with a large, circular initial 'J' and a long horizontal flourish extending to the right.

Fredsvej 11, DK-2840 Holte, Denmark – January 21, 2010