32 Years of VDM
From Earliest Days via Adolescence to Maturity
Dines Bjørner

Computer Science and Engineering
Informatics and Mathematical Modelling
Technical University of Denmark
DK-28000 Kgs.Lyngby
Denmark

Graduate School of Information Science
Japan Adv. Inst. of Science & Technology
1-1, Asahidai, Tatsunokuchi
Nomi, Ishikawa 923-1292
Japan

bjorner@gmail.com


Overview

● We survey a history of VDM

★ from its inception at the IBM Vienna Laboratory during the period of May 1973 through early 1975,
★ via its ”growing up” period between the mid 1970s and into the early 1980s
  ◇ at The Technical University of Denmark (DB, 1976 – ...)
  ◇ at IBM’s European Systems Research Institute (Cliff Jones, 1976 – 1978),
  ◇ and at Oxford and Manchester Univs. (UK, Cliff Jones, 1978 – ...),
★ to its beginning
  ◇ industrial acceptance,
  ◇ tool building,
  ◇ through VDM Europe (now the Formal Methods Europe)
  ◇ and BSI and later ISO standardisation
★ reaching a current industrial mile-stone through the wider spread of VDM by the Japanese company CSK Group.
In-between we relate major uses of VDM in early world-wide projects:

- the formal semantics of the C.C.I.T.T. (now ITU) programming language CHILL,
- the development of a portable (the only) compiler for full CHILL,
- the development of the commercially most successful compiler for Ada,
- the Formal Definition of Ada - where also other formalisms were used,
- and so on.
• We end the tutorial with speculations on the acceptance and propagation of formal methods like VDM in academia and in industry.

• The talk will be peppered by brief, one-slide example of formalisations in VDM.

• The aim of the tutorial is to inform.

• The objective of the tutorial is for the participant to understand
  ★ how formal methods evolve,
  ★ their acceptance or lack thereof,
  ★ their aspirations and failures,
  ★ what drives certain researchers towards study of formal methods,
  ★ and what initially drives some software engineers cum technologists towards uses of formal methods.
1st VDM Example: Aircraft Tracking

LAT = Real
  \text{inv} \; \text{lat} == \; \text{lat} \geq 0 \; \text{and} \; \text{lat} < 360
LON = Real
  \text{inv} \; \text{lon} == \; \text{lon} \geq -90 \; \text{and} \; \text{lon} \leq +90
ALT = Real
  \text{inv} \; \text{alt} \geq -340

\text{Air\_Cra\_Id} = \text{token}
\text{Air\_Cra\_Pos :: lat:\text{LAT} \; lon:\text{LON} \; alt:\text{ALT}}

\text{RadarInfo} = \text{map} \; \text{Air\_Cra\_Id to Air\_Cra\_Pos}
  \text{inv} \; \text{ri} == \; // \; \text{no two distinct aci have same position}
A VDM History
Conception, Birth and Baby: 1973 – 1975

• The spiritual ancestor of VDM was VDL: The Vienna Definition Language:
  ★ VDL was nicknamed so by J.A.N. Lee, mid 1970s
  ★ VDL was used in defining the semantics of PL/I, 1964 – 1969
  ★ at the IBM Vienna Laboratory (Vienna, Austria, 1961 – 1999)

• VDM was
  ★ conceived around May 1973
  ★ and born and christened in September 1974
  ★ in connection with the production development of a PL/I compiler
    for the “new” series of IBM computers: FSM (“future systems machines”)\(^1\)

\(^1\)The IBM FSM project was abandoned in Feb. 1974
• The intellectual parents of VDM were:
  ★ **Hans Bekič** († 1982),
  ★ Wolfgang Henhapl,
  ★ **Cliff Jones (CBJ),**
  ★ **Peter Lucas** and
  ★ **me.**

• VDM was then used in a number of projects
  ★ at the IBM Vienna Lab. till mid 1975
  ★ but “grew” mostly outside IBM!

• **A first lesson:**
  ★ If IBM had kept “supporting” VDM then VDM would have died!
  ★ Fresh air, peer review/critique of academia is healthy.²

²Now IBM has embraced UML! With friends like that who needs enemies?
2nd VDM Example: Programming Language Storage

VAL = ScaVAL | ComVAL
ScaVAL = IntgVAL | BoolVAL | CharVAL
ComVAL = RecVAL | ArrVAL
IntgVAL :: iv:Int, BoolVAL :: bv:Bool, CharVAL :: cv:Char
RecVAL :: rv:map Fid to VAL
ArrVAL :: av:map seq of Intg to VAL

LOC = ScaLOC | ComLOC
ScaLOC = IntgLOC | BoolLOC | CharLOC
ComLOC = RecLOC | ArrLOC
IntgLOC :: il:TOKEN, bl:BoolLOC :: TOKEN, CharLOC :: cl:TOKEN
RecLOC :: rl:map Fid to LOC
ArrLOC :: al:map seq of Intg to LOC

1STG = map ScaLOC to ScaVAL  // Algol 60 //
rSTG = map TOKEN to VAL  // Pascal //
STG = map LOC to VAL  // PL/I, Ada, ... //
Toddler: 1976 – 1980s

- Three strands:
  - The **"Danish" School** (DB):
    1. Copenhagen University (Sept. ’75 – Aug. ’76)
    2. Technical University of Denmark (1976 – …)
  - The **"British" School** (CBJ):
    1. The IBM ESRI, mid 1970s
    2. Oxford University, late 1970s
    3. Manchester University, 1980 – …
  - The **"Irish" School** (MMaA): 1980s – …

- Joint “propagation”:
  - Springer LNCS Vol. 61 (DB + CBJ)
  - The Lyngby (DK) Winter School, Jan. 1979: First real “launch” of VDM
  - Springer LNCS Vol.86 (DB + CBJ)

---

This 2 week school had 117 attendants from east and west Europe and featured lectures also by J.E. Stoy (Den.Sem.), S.N. Zilles & B. Liskov (Alg.Sem.), R.M. Burstall (Clear), P. Lauer, G.D. Plotkin (SOS), O.-J. Dahl — besides CBJ + DB
Danish (DDC) VDM and VDM-derivative Projects 1980s

- **Formal Description of Ada** (J. Bundgaard, O. Dommergaard, H. H. Løvengreen, J. S. Pedersen, L. Schultz; eds. DB + O. N. Oest), Springer LNCS Vol. 98, 1980
- **CHILL Compiler** development, 1981–1984, DDCI Inc. (S. Prehn, P. L. Haff)
- **Ada Compiler** development, 1981–1984, DDCI Inc. (EU sponsored)
- Four–five Danish industry projects using VDM (@ DDC)
- **Formal Definition of Ada** (EU sponsored) Static Semantics was expressed in VDM. DTU, DDC, Univ. of Genoa, ...
- **FMA: Formal Methods Appraisal** (EU sponsored) DTU, DDC, STL (UK), ...
- **RAISE**: Rigorous Approach to Industrial Software Engineering
  A spec.lang., method and tool project (EU sponsored) DTU, DDC, STL (UK), Bull (F), ...
- **LaCOS**: Large Software Devt. using Formal Methods — RAISE (EU sponsored)
  DDC, STL (UK), Bull (F), Matra (D), Technisystems (GR), Space Systems (It), Espacio (ES), ...
3rd VDM Example: Sets

\[ G = \text{set of } C \]  // type: group of citizens

let \( g = \{c_1, c_2, \ldots, c_n\} \) in \( f(g) \)  // defining a group

\( g = g' \)  // groups \( g \) and \( g' \) have same citizens
\( g <> g' \)  // groups \( g \) and \( g' \) do not have same citizens
\( g \text{ subset } g' \)  // group \( g \) is contain in group \( g' \)
c \text{ in set } g  // is citizen \( c \) in group \( g \) ?
c \text{ not in set } g  // is citizen \( c \) not in group \( g \) ?

\( g \text{ union } g' \)  // merge of two groups
\( g \text{ inter } g' \)  // citizens common to two groups
\( g \setminus g' \)  // citizens in \( g \) but not in \( g' \)

\( \text{card } g \)  // number of citizens in group \( g \)
\( \text{dunion } \{g, g', \ldots, g''\} \)  // group of citizens in any group
\( \text{dinter } \{g, g', \ldots, g''\} \)  // group of citizens in all groups
British VDM and VDM-derivative Projects 1980–90s

- Cliff Jones books

- and projects:
  - μral (Manchester University / Rutherford Appleton Lab.)
    Proof assistant for VDM development

- Other books:
  - Proof in VDM: a practitioner’s guide, 1994
    J.C. Bicarregui, J.S. Fitzgerald, P.A. Lindsay, R. Moore
4th VDM Example: Cartesians (Records)

Day == \(<\text{Mo}>|<\text{Tu}>|<\text{We}>|<\text{Th}>|<\text{Fr}>|<\text{Sa}>|<\text{Su}>\)
Mon == \(<\text{Jan}>|<\text{Feb}>|<\text{Mar}>|<\text{Apr}>|<\text{May}>|<\text{Jun}>|<\text{Jul}>|<\text{Aug}>|<\text{Sep}>|<\text{Oct}>|<\text{Nov}>|<\text{Dec}>\)

Year = \text{Nat}

Date :: day:Day month:Month year:Year

\textbf{let} date = \textbf{mk}Date(Fr,Oct,2006) \textbf{in} ... // \textbf{mk}Date: constructor

date.day = Fr // selector day
date.month = Oct // selector month
date.year = 2006 // selector year

date = date’
date <> date’’
5th VDM Example: Alternative Records, _is_ Functions

Exp = Num | Ide | Pre | Inf | IfTh | Appl
Num :: Real
Ide :: Token
Pre :: po:Pop ex:Exp
Inf :: lex:Exp io:Iop rex:Exp
IfT :: if:Exp th:Exp el:Exp
Apl :: fct:Fnm arg:Exp

Pop == <\text{min}|<\text{fac}>
Iop == <\text{min}|<\text{plu}|<\text{mpy}|<\text{idiv}|<\text{gre}|<\text{geq}|<\text{equ}|<\text{neq}|<\text{sma}|<\text{seq}|...

7 + (\text{if } j=3 \text{ then } -5 \text{ else } \text{square}(2))

\text{mkInf(mkNum(7),plu,mkIfT(mkInf(mkIde(j),equ,mkNum(3))), mkPre(min,mkNum(5)), mkApl(mkIde(square),mkNum(2)))}}
6th VDM Example: *is_* Functions

\[
\begin{align*}
\text{Exp} &= \text{Num} \mid \text{Ide} \mid \text{Pre} \mid \text{Inf} \mid \text{IfTh} \mid \text{Appl} \\
\text{Num} &:: \textbf{Real} \\
\text{Ide} &:: \textbf{Token} \\
\text{Pre} &:: \text{po} : \text{Pop} \ \text{ex} : \text{Exp} \\
\text{Inf} &:: \text{lex} : \text{Exp} \ \text{io} : \text{Iop} \ \text{rex} : \text{Exp} \\
\text{IfT} &:: \text{if} : \text{Exp} \ \text{th} : \text{Exp} \ \text{el} : \text{Exp} \\
\text{Apl} &:: \text{fct} : \text{Fnm} \ \text{arg} : \text{Exp} \\
\end{align*}
\]

\[
\begin{align*}
\text{is}_\text{Num}: \text{Exp} &\rightarrow \textbf{Bool} \\
\text{is}_\text{Ide}: \text{Exp} &\rightarrow \textbf{Bool} \\
\text{is}_\text{Pre}: \text{Exp} &\rightarrow \textbf{Bool} \\
\text{is}_\text{Inf}: \text{Exp} &\rightarrow \textbf{Bool} \\
\text{is}_\text{IfT}: \text{Exp} &\rightarrow \textbf{Bool} \\
\text{is}_\text{Fnm}: \text{Exp} &\rightarrow \textbf{Bool}
\end{align*}
\]
7th VDM Example: Semantics of Expressions

Syntactic Types:

\[ \text{Exp} = \text{Num} \mid \text{Ide} \mid \text{Pre} \mid \text{Inf} \mid \text{IfTh} \mid \text{Apl} \]

- \text{Num} :: \text{Real}
- \text{Ide} :: \text{Token}
- \text{Pre} :: \text{po:Pop} \mid \text{ex:Exp}
- \text{Inf} :: \text{lex:Exp} \mid \text{io:Iop} \mid \text{rex:Exp}
- \text{IfT} :: \text{if:Exp} \mid \text{th:Exp} \mid \text{el:Exp}
- \text{Apl} :: \text{fct:Fnm} \mid \text{arg:Exp}

Semantic Types:

- \text{FCT} = \text{VAL} \rightarrow \text{VAL}
- \text{ENV} = \text{map} \text{ Ide} \text{ to } (\text{LOC} \mid \text{FCT})
- \text{STG} = \text{map} \text{ LOC} \text{ to } \text{VAL}

E: \text{Exp} \rightarrow \text{ENV} \rightarrow \text{STG} \rightarrow \text{VAL}

\[ E(e)_{\rho\sigma} \rightarrow \text{cases } e: \]

- \text{mkNum}(r) \rightarrow n,
- \text{mkIde}(t) \rightarrow \sigma(\rho(e)),
- \text{mkPre}(o,e') \rightarrow M(o)(E(e')_{\rho\sigma}),
- \text{mkInf}(le,o,re) \rightarrow M(o)(E(le)_{\rho\sigma},E(re)_{\rho\sigma})
- \text{mkIfT}(b,c,a) \rightarrow \text{if } E(b)_{\rho\sigma}
  \begin{align*}
  & \text{then } E(c)_{\rho\sigma} \\
  & \text{else } E(a)_{\rho\sigma}
  \end{align*}
- \text{mkApl}(f,e') \rightarrow \rho(f)(E(e')_{\rho\sigma})

end
VDM Standardisation

- Reasons for standardisation:
  1. control of syntax and confirmation of semantics,
  2. recognition — academia and industry,
  3. support for industry.
  - Chair: Derek Andrews, UK
  - Denmark: Dines Bjørner, Bo Stig Hansen, Peter Gorm Larsen, et al.,
  - France: Jean Goubault, Patrick Behm, et al.,
  - Germany: Kiel Univ. staff
  - Ireland: Mícheál Mac an Airchinnigh, Andrew Butterfield, et al.,
  - Japan: Makoto Someya, Yamamura Yoshinobu, et al.,
  - The Netherlands: Nico Plat, Hans Toetenel, et al.,
  - Poland: Andrzej Blikle, Wiesiek Pawlowski,
  - UK: Cliff Jones, John Dawes, Brian Q. Monahan, Roger Scowen, J.S. Fitzgerald et al.
- http://www.vdmportal.org/
8th VDM Example: Sequences

```
seq of B

let ℓ = [b1,b2,...,bm] in ...

hd ℓ       // first element of a list
tl ℓ        // list of all but first element
len ℓ       // length of a list
elems ℓ     // set of distinct element of a list
inds ℓ      // set of list indices: 1..len ℓ
ℓ' ~ ℓ''    // concatenation of two lists
conc[ℓ₁, ℓ₂, ..., ℓₘ] ≡ ℓ₁ ~ ℓ₂ ~ ... ~ ℓₘ
ℓ' = ℓ''     // equal lists
ℓ' <> ℓ''    // unequal lists
ℓ(i)         // selection of i’th list element
```
Danish – IFAD – VDM and VDM-derivative Projects 1990-2000s

- The mastermind: Peter Gorm Larsen
- The institution: IFAD, Inst. For Applied Datalogy (Comp.Cci.)
- The results:
  - Several MScs, PhDs and visitors (@ IFAD) — also from Japan!
  - The VDM Toolbox

- Many IFAD industry projects, in Europe and in the US:
  Boeing, British Aerospace, Aerospatiale, Dassault, Matra, Alcatel and Baan.

9th VDM Example: Maps

\[ \text{DIR} = \text{map \ Did to (FILE | DIR)} \]

\textbf{let} \ dir = \{id1 \leftrightarrow f1, id2 \leftrightarrow dir2, \ldots, idn \leftrightarrow fn\} \textbf{in} \ldots

\textbf{dom} dir \hspace{1cm} // set of 1st level directory identifiers

\textbf{rng} dir \hspace{1cm} // set of 1st level directory files and directories

\textbf{dir1 union dir2} \hspace{1cm} // merge of two identifier disjoint directories

\textbf{dir1 ++ dir2} \hspace{1cm} // overriding 1st directory by 2nd

\textbf{merge dirs} \hspace{1cm} // merge of set of disjoint directories

\textbf{s <: dir} \hspace{1cm} // directory restricted to identifiers of s

\textbf{s <<- dir} \hspace{1cm} // directory restricted to identifiers not in s

\textbf{dir :> s} \hspace{1cm} // range restricted to identifiers of s

\textbf{dir :-> s} \hspace{1cm} // range restricted to identifiers not in s

\textbf{dir1 = dir2} \hspace{1cm} // equal directories

\textbf{dir1 <> dir2} \hspace{1cm} // different directories
VDM Cultures

1. IBM Vienna Lab. 1960s – 1975
2. Dansk Datamatik Center 1979 – 1989
3. Manchester University 1980s
4. IFAD 1990s
5. University of Newcastle late 1990s – ...
6. IHA 2000s – ...
7. CSK, Japan 2000s – ...

Other “strongholds”:
- Trinity College, Ireland
- Adelard, UK
- Schleschwig Holstein, Germany
- TU Delft, The Netherlands
- University of Minho, Portugal
- Etcetera

---

4The Engineering University College of Århus: P.G. Larsen
10th VDM Example: Function Definitions (Graphs)

\[ G = \text{map } N \to \text{set of } N \]

let \( g = \{ n_1 \mapsto \{n_2, n_3\}, n_2 \mapsto \{n_1\}, n_3 \mapsto \{n_4, n_2\}, n_4 \mapsto \{\}, n_5 \mapsto \{\} \} \) in ...

\text{is\_node\_in\_graph}: N \times G \to \text{Bool}
\text{is\_edge\_in\_graph}: (N \times N) \times G \to \text{Bool}
\text{insert\_node}: N \times G \to G
\text{delete\_node}: N \times G \to G
\text{insert\_edge}: (N \times N) \times G \to G
\text{delete\_edge}: (N \times N) \times G \to G

\text{is\_node\_in\_graph}(n, g) == n \text{ in set dom } g
\text{is\_edge\_in\_graph}((n_1, n_2), g) == \text{is\_node\_in\_graph}(n_1, g) \text{ and } n_2 \text{ in set rang } g(n_1)
\text{insert\_node}(n, g) == g \text{ union } \{n \mapsto \{\}\}
\text{delete\_node}(n, g) == g <\cdot: \{n\}
\text{insert\_edge}((n_1, n_2), g) == g ++ \{n_1 \mapsto g(n_1) \text{ union } \{n_2\}\}
\text{delete\_edge}((n_1, n_2), g) == g ++ \{n_1 \mapsto g(n_1) \setminus \{n_2\}\}
Analysing Specifications: Gaining Further Trust

- A major issue of abstract specifications is understanding.
  - Abstracting a design gives insight.
  - One can focus on the essential properties of an abstract design.
  - In steps of development — see next — one can then achieve concrete efficiency.
- But even an abstract design may be fallacious, i.e., have “errors”.
- We therefore need analyse the abstractions.
11th VDM Example: Proof Obligations

Basis

pre insert_node(n,g): not is_node_in_graph(n,g)
pre delete_node(n,g): is_node_in_graph(n,g) and g(n)=\{
pre insert_edge((n1,n2),g): \{n1,n2\} subset dom g and not is_edge_in_graph((n1,n2),g)
pre delete_edge((n1,n2),g): \{n1,n2\} subset dom g and is_edge_in_graph((n1,n2),g)

Obligations

• For any function definition and for any use of above functions
• it must be shown that the pre-conditions hold.
Obligations (Continued)

- Further it must be shown that graphs resulting from the above
- satisfy the graph invariant.

12th VDM Example: Invariants

\[
G = \text{map } N \text{ to set of } N \\
\text{inv } g \implies \text{dom } g = \text{dunion } \text{rng } g
\]
Stepwise Development

- A main, an almost overriding issue of software development is this:
  - ★ Start with an “as abstract” a model,
    ◦ but also a “no more abstract” model than necessary.
  - ★ That model is usually not “immediately executable”.
  - ★ Fine!
  - ★ Then refine the abstract model to a less abstract model.
  - ★ Gradually introducing “efficiency” measures (storage and time).
  - ★ Eventually reach a model which is “immediately executable”.
  - ★ That is, which can be “believably” transliterated into f.ex. Java or C# or ...
  - ★ Prove every step of refinement.
Abstractly graphs are maps from nodes to sets of nodes.

Concretely graphs may be represented as a pointer structure in storage:

- A (possibly empty) linked list, the **node chain**, of node chain records
- To which is attached a (possibly empty) linked list, the **adjacency chain**, of adjacency node records.

We need to establish a believable stepwise refinement from the former to the latter.
14th VDM Example: Graphs (Continued)

\[ G_0 = \text{map } N \text{ to set of } N \]
\[ g_0: \{a\rightarrow\{b\},b\rightarrow\{c,d\},c\rightarrow\{c,d,e\},e\rightarrow\{\},d\rightarrow\{a\}\} \]

\[ G_1 = \text{set of } (N \times \text{set of } N) \]
\[ g_1: \{(a,\{b\}),(b,\{c,d\}),(c,\{c,d,e\}),(e,\{}),(d,\{a\})\} \]

\[ G_2 = \text{seq of } (N \times \text{seq of } N) \]
\[ g_1: \[(a, [b]), (b, [c,d]), (c, [c,d,e]), (e, []), (d, [a])\] \]

\[ G_3 = N \times \text{NChain} \times \text{AChain} \]
\[ \text{NChain} = \text{map } \text{NPtr} \text{ to } \text{NRec} \]
\[ \text{AChain} = \text{map } \text{APtr} \text{ to } \text{ARec} \]
\[ \text{NRec} = (\text{NPtr}|\text{nil}) \times N \times (\text{APtr}|\text{nil}) \]
\[ \text{ARRec} = \text{NPtr} \times (\text{APtr}|\text{nil}) \]

\[ N, \text{NPtr, APtr} = \text{Token} \]
Features of VDM–SL Not Mentioned

• Modularisation
• LPF: Logic for Partial Functions
• Scalar types: real, integer, characters, boolean
• Imperative programming: variables, assignments, statements, ...

Please Study the Books
The Triptych Dogma of Software Engineering

Motivation

• Before software can be designed, even abstractly, as shown above,
  • we must understand the requirements.

• So the requirements prescription must likewise be formalised.

• But before we can formalise the requirements
  • we must understand the application domain
    in which the software is to reside.

• So the domain description must likewise be formalised.
The Dogma

- Software engineering proceeds, ideally, as follows:

  ★ **Domain engineering:**
  - Acquiring, analysing and modelling the domain,
  - as it is, with no reference to requirements, let alone software,
  - informally and formally – verifying and validating this model.

  ★ **Requirements engineering:**
  - Transforming, informally, the domain description model,
  - in stages of development into a requirements prescription model,
  - informally and formally – verifying and validating this model.

  ★ **Software design:**
  - Finally transforming, formally, the requirements prescription model
  - in stages and steps of development (refinement) into a software design,
  - verifying correctness of this design wrt. requirements in the context of the domain model:

\[
D, S \models R
\]
Domain Engineering: The New Thing

- In classical engineering branches engineers build on theories of physics and mathematics.

- Now, in software engineering you will be asked to build on the sciences of the domains, i.e., on domain theories and on computer & computing science.

- Today SEs develop software for
  - hospitals, railways, banks, manufacturing, the web,
  - without first having clear descriptions of these domains.

- Not so in future.
Obstacles / Hindrances to Professionalism

• It is **professional** for an aeronautics engineer **to know** his domain: aerodynamics.

• It is **not professional** for a software engineer **to not know** the domain: hospitals, or railways, or banks, or mfg., ...

• Which are the **reasons** for this **sad state-of-affair**?
  
  ★ **Lack of awareness:** ...  
  ★ **Absence of critical mass:** ...  
  ★ **Generation gap:** ...  
  ★ **Customers are unaware:** ...  
  ★ **Our colleagues are unaware:** ...

---

5 In Europe most univs. teach FMs.
6 In most software houses young candidates knowledgeable in FMS are put to work in groups most of whose members do not know FMS.
7 Most software house managers are not professionally educated.
8 If they knew they would demand use of FMs — thus obtaining trustworthy software.
9 Students also take courses from colleagues who are not properly educated. Instead they use “antiquated math.” to deal with computational models.
From VDM to RAISE: Rigorous Approach to Industrial SE

- VDM was first conceived in 1973.
- By 1984 a number of shortcomings were identified in an EU sponsored project between DDC (Denmark) and STL (UK).
- The result was the likewise EU sponsored RAISE project: 1985-1989.
- RAISE builds on:
  - VDM
  - OBJ
  - CSP
  - “Strong” typing
- Thus the RAISE Specification Language, RSL allows for:
  - sorts, observer and generator functions, and axioms,
  - parameterised modularisation
    in a different, possibly more general way than does VDM,
  - concurrency,
  - and all that VDM can express.
Study the Springer Books

1. D. Bjørner: *Software Engineering*,
   *Vol. 1: Abstraction and Modelling* (Jan., 2006)

2. D. Bjørner: *Software Engineering*,

   *Vol. 3: Domains, Requirements and Software Design* (March, 2006)

2414 pages, almost 6,000 lecture slides!
Conclusion

- We have surveyed a biased history of VDM
- And we have presented a number of VDM examples
- We have advocated the use of formal methods (FM)s
- And we have enlarged the scope of use of FM}s from software design to also include domains and requirements
- We have “speculated” on the propagation of FM}s
- Finally we have briefly mentioned RAISE, a follow-on to VDM

THANKS