32 Years of VDM
From Earliest Days via Adolescence to Maturity

Dines Bjørner

Computer Science and Engineering	Graduate School of Information Science
Informatics and Mathematical Modelling	Japan Adv. Inst. of Science & Technology
Technical University of Denmark	1-1, Asahidai, Tatsunokuchi
DK-28000 Kgs.Lyngby	Nomi, Ishikawa 923-1292
Denmark	Japan

bjorner@gmail.com

June 14, 2006. Compiled October 29, 2006: 14:08

[0]Invited tutorial for the IPSJ/SIGSE Software Engineering Symposium 2006, Friday 21 October 2006, Tokyo, Japan (SIGSE: Special Interest Group of Software Engineering, IPSJ: Information Processing Society of Japan). Venue: Koto-ku, Tokyo, Miraikan (National Museum of Engineering Science and Innovation)

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**
  inv lat == lat $\geq$ 0 **and** lat<360
LON = **Real**
  inv lon == lon ≥ -90 **and** lon $\leq +90$
ALT = **Real**
  inv alt ≥ -340

Air_Cra_Id = token
Air_Cra_Pos :: lat:LAT lon:LON alt:ALT

RadarInfo = **map** Air_Cra_Id **to** Air_Cra_Pos
  inv ri == // 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]

[2]Now 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 – ...)
    3. Dansk Datamatik Center (**DDC,** 1979–1989)
  - ⋆ 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[3]
  - ⋆ Springer LNCS Vol.86 (DB + CBJ)
  - ⋆ The Prentice Hall Intl. book: 1982 (DB + CBJ)

[3] 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
- **Semantics of CHILL** (H.Bruun, P.L. Haff) The C.C.I.T.T. Standard, 1978–1982
- **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 = **set of** C	// type: group of citizens
let g = {c1,c2,...,cn} **in** f(g)	// defining a group
g = g′	// groups g and g′ have same citizens
g <> g′	// groups g and g′ dos not have same citizens
g **subset** g′	// group g is contain in group g′
c **in set** g	// is citizen ci in group g ?
c **not in set** g	// is citizen ci not in group g ?
g **union** g′	// merge of two groups
g **inter** g′	// citizens common to two groups
g \ g′	// citizens in g but not in g′
card g	// number of citizens in group g
dunion {g,g′,...,g″}	// group of citizens in any group
dinter {g,g′,...,g″}	// group of citizens in all groups

British VDM and VDM-derivative Projects 1980–90s

- Cliff Jones books

  1. Software Development: A Rigorous Approach, 1980
  2. Formal Specification and Software Development, (w/ DB) 1982
  3. Systematic Software Development Using VDM, 1986
  4. Case Studies in Systematic Software Development, (w/ R.C.F.Shaw) 1989
  5. Systematic Software Development using VDM (2nd Edition), 1990
  6. MURAL: A Formal Development Support System, (w/ et al.), 1991

- and projects:

  ⋆ μral (Manchester University / Rutherford Appleton Lab.)
  Proof assistant for VDM development

- Other books:

  ⋆ The VDM-SL Reference Guide, J. Dawes, 1991

  ⋆ 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 == <Mo>|<Tu>|<We>|<Th>|<Fr>|<Sa>|<Su>
Mon == <Jan>|<Feb>|<Mar>|<Apr>|<May>|<Jun>|
<Jul>|<Aug>|<Sep>|<Oct>|<Nov>|<Dec>
Year = **Nat**
Date :: day:Day month:Month year:Year

let date = **mk**Date(Fr,Oct,2006) **in** ... // **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 == <min>|<fac>
Iop == <min>|<plu>|<mpy>|<idiv>|<gre>|<geq>|<equ>|<neq>|<sma>|<seq>|...

7 + (**if** *j=3* **then** *-5* **else** *square(2)*)

mkInf(**mk**Num(7),plu,**mk**IfT(**mk**Inf(**mk**Ide(j),equ,**mk**Num(3)),
mkPre(min,**mk**Num(5)),
mkApl(**mk**Ide(square),**mk**Num(2))))

6th VDM Example: *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

is_Num: Exp → **Bool**
is_Ide: Exp → **Bool**
is_Pre: Exp → **Bool**
is_Inf: Exp → **Bool**
is_IfT: Exp → **Bool**
is_Fnm: Exp → **Bool**

7th VDM Example: Semantics of Expressions

Syntactic Types:

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

Semantic Types:

FCT = VAL → VAL
ENV = **map** Ide **to** (LOC|FCT)
STG = **map** LOC **to** VAL

E: Exp → ENV → STG → VAL
E(e)$\rho\sigma$ ==
  cases e:
    mkNum(r) → n,
    mkIde(t) → $\sigma(\rho(\text{e}))$,
    mkPre(o,e′) → M(o)(E(e′)$\rho\sigma$),
    mkInf(le,o,re) → M(o)(E(le)$\rho\sigma$,E(re)$\rho\sigma$)
    mkIfT(b,c,a) → **if** E(b)$\rho\sigma$
                then E(c)$\rho\sigma$
                else E(a)$\rho\sigma$
    mkApl(f,e′) → ρ(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.
- The BSI/ISO VDM-SL effort: 1987–1996 (JTC1/SC22/WG19/)
  - ⋆ 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
$\ell' \frown \ell''$ // concatenation of two lists
conc $[\ell_1, \ell_2, ..., \ell_m] \equiv \ell_1 \frown \ell_2 \frown ... \frown \ell_m$
ℓ'=ℓ'' // equal lists
ℓ'<>ℓ'' // unequal lists
ℓ(i) // selection of i'th list element

anish – 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**
  - ⋆ The book: **Modelling Systems: Practical Tools and Techniques, J.S. Fitzgerald** and P.G. Larsen, Cambridge University Press, 1998 also in Japanese
- Many **IFAD** industry projects, in Europe and in the US: Boeing, British Aerospace, Aerospatiale, Dassault, Matra, Alcatel and Baan.
- The book: **Validated Designs for Object-oriented Systems, J.S. Fitzgerald**, P.G. Larsen, P. Mukherjee, N. Plat, M. Verhoef, Springer, 2005

9th VDM Example: Maps

DIR = **map** Did **to** (FILE | DIR)

let dir = {id1 ↦ f1,id2 ↦ dir2, ..., idn ↦ fn} **in** ...

dom dir	// set of 1st level directory identifiers
rng dir	// set of 1st level directory files and directories
dir1 **union** dir2	// merge of two identifier disjoint directories
dir1 ++ dir2	// overriding 1st directory by 2nd
merge dirs	// merge of set of disjoint directories
s <: dir	// directory restricted to identifiers of s
s <-: dir	// directory restricted to identifiers not in s
dir :> s	// range restricted to identifiers of s
dir :-> s	// range restricted to identifiers not in s
dir1 = dir2	// equal directories
dir1 <> dir2	// 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**[4]	2000s – ...
7. **CSK, Japan**	2000s – ...

Other "strongholds":

- Trinity College, Ireland
- Adelard, UK
- Schleschwig Holstein, Germany
- TU Delft, The Netherlands
- University of Minho, Portugal
- $\mathcal{E}$tcetera

[4] The Engineering University College of Århus: P.G. Larsen

10th VDM Example: Function Definitions (Graphs)

G = **map** N **to** **set of** N

let g = {n1 ↦ {n2,n3},n2 ↦ {n1}, n3 ↦ {n4,n2}, n4 ↦ {}, n5 ↦ {}} **in** ...

is_node_in_graph: N ∗ G → **Bool**
is_edge_in_graph: (N ∗ N) ∗ G → **Bool**
insert_node: N ∗ G → G
delete_node: N ∗ G → G
insert_edge: (N ∗ N) ∗ G → G
delete_edge: (N ∗ N) ∗ G → G

is_node_in_graph(n,g) == n **in set dom** g
is_edge_in_graph((n1,n2),g) == is_node_in_graph(n1,g) **and** n2 **in set rang** g(n1)
insert_node(n,g) == g **union** {n ↦ {}}
delete_node(n,g) == g <−: {n}
insert_edge((n1,n2),g) == g ++ {n1 ↦ g(n1) **union** {n2}}
delete_edge((n1,n2),g) == g ++ {n1 ↦ g(n1) \ {n2}}

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

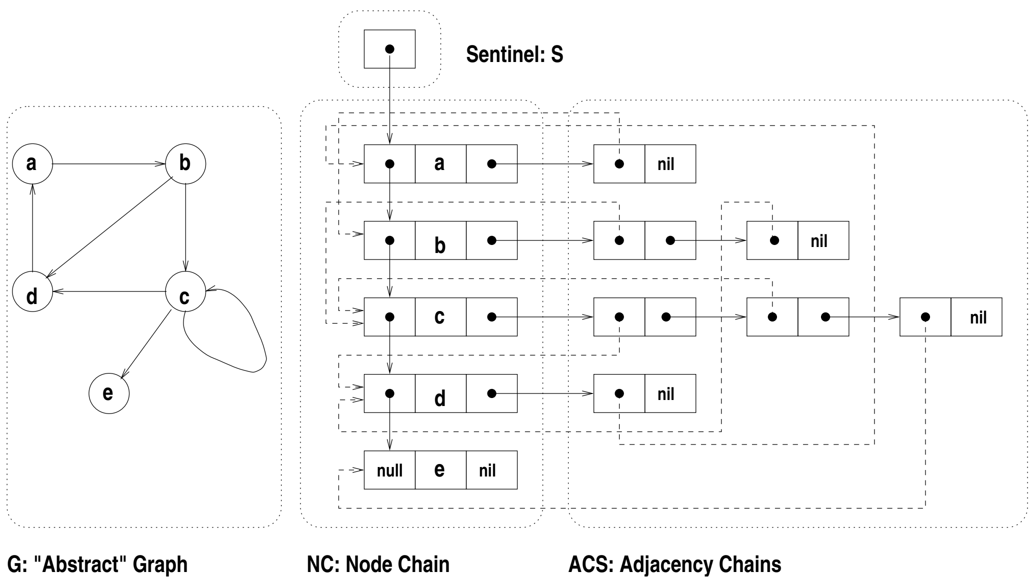
G = **map** N **to set of** N
inv g == **dom** g = **dunion rng** g

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.**

13th VDM Example: Graphs

- 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)

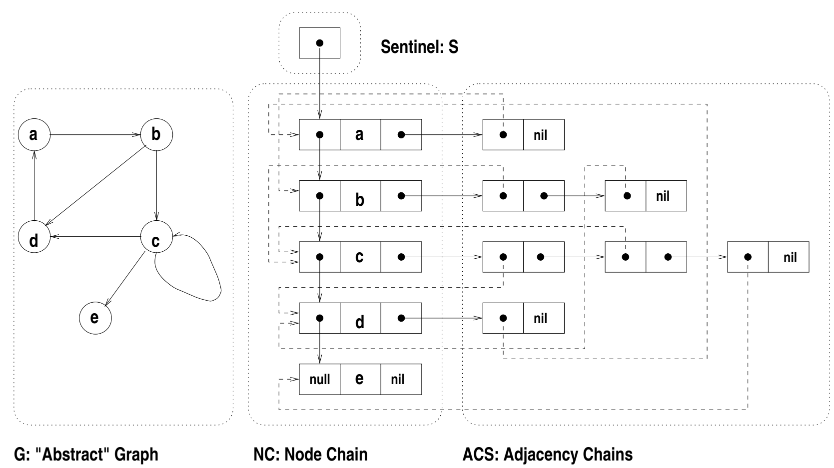
G0 = **map** N **to set of** N
g0: {a↦{b},b↦{c,d},c↦{c,d,e},e↦{},d↦{a}}

G1 = **set of** (N ∗ **set of** N)
g1: {(a,{b}),(b,{c,d}),(c,{c,d,e}),(e,{}),(d,{a})}

G2 = **seq of** (N ∗ **seq of** N)
g1: [(a,[b]),(b,[c,d]),(c,[c,d,e]),(e,[]),(d,[a])]

G3 = N ∗ NChain ∗ AChain
NChain = **map** NPtr **to** NRec
AChain = **map** APtr **to** ARec
NRec = (NPtr|**nil**) ∗ N ∗ (APtr|**nil**)
ARec = NPtr ∗ (APtr|**nil**)

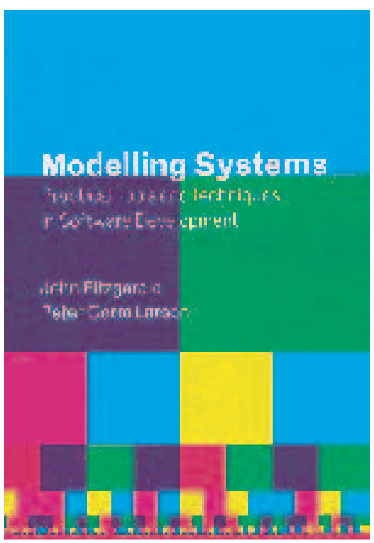
N, NPtr, APtr = Token

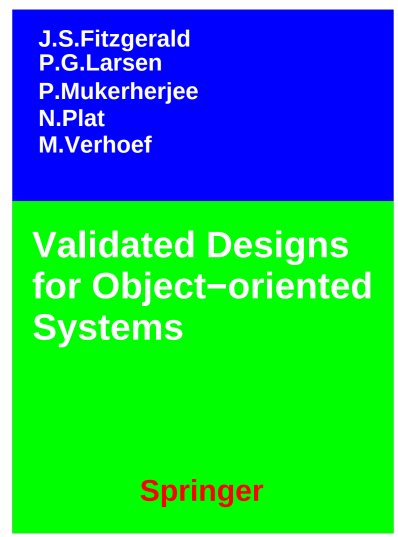


Features of VDM–SL Not Mentioned

- 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 **design**ed, 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:
  - ⋆ $\mathcal{D}$**omain 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.
  - ⋆ $\mathcal{R}$**equirements engineering:**
    - ◇ Transforming, informally, the domain description model,
    - ◇ in stages of development into a requirements prescription model,
    - ◇ informally and formally – verifying and validating this model.
  - ⋆ $\mathcal{S}$**oftware 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:

$$\mathcal{D}, \mathcal{S} \models \mathcal{R}$$

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:** ...[5]
  - ⋆ **Absence of critical mass:** ...[6]
  - ⋆ **Generation gap:** ...[7]
  - ⋆ **Customers are unaware:** ...[8]
  - ⋆ **Our colleagues are unaware:** ...[9]

[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.

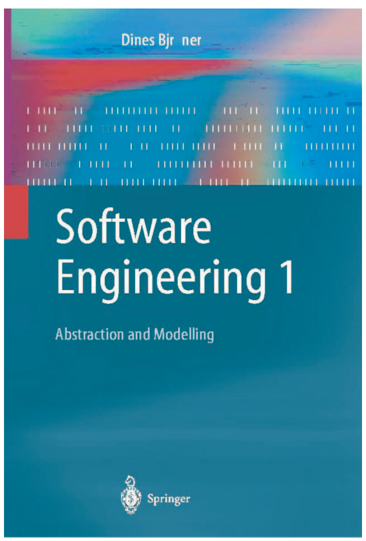
From VDM to RAISE: Rigorous Approach to Industrial SE

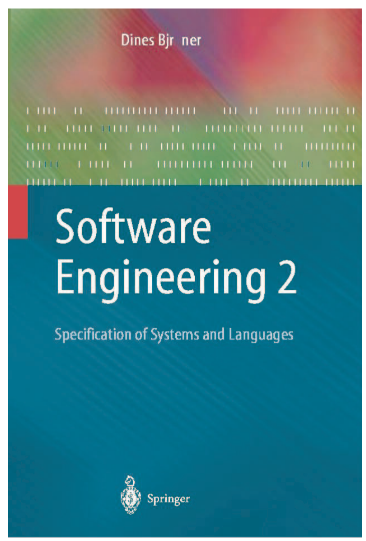
- VDM was first conceived in 1973.
- By 1984 a number of shortcomings were identified in an EU sponsored project between DDC (Denmark) and STL (UK).
- The result was the likewise EU sponsored RAISE project: 1985-1989.
- RAISE builds on:
  - ⋆ **VDM** ⋆ **OBJ** ⋆ **CSP** ⋆ **"Strong" typing**
- Thus the **RAISE S**pecification **L**anguage, **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.

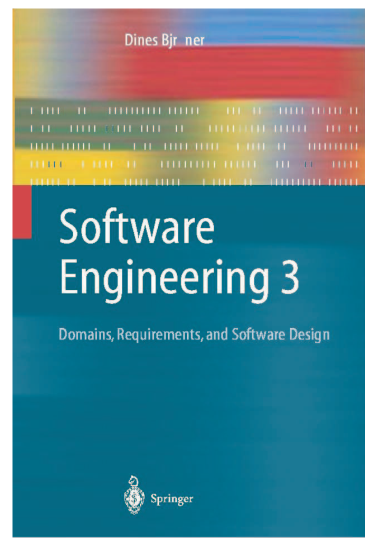
Study the Springer Books

1. D. Bjørner: ***Software Engineering***,
   Vol. 1: ***Abstraction and Modelling*** (Jan., 2006)
2. D. Bjørner: ***Software Engineering***,
   Vol. 2: ***Specification of Systems and Languages*** (Feb., 2006)
3. 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 (FMs)
- And we have enlarged the scope of use of FMs
  from software design to also include domains and requirements
- We have "speculated" on the propagation of FMs
- Finally we have briefly mentioned RAISE, a follow-on to VDM

THANKS