Specifications of IT systems – Formal specification techniques including VDM-SL modelling

Peter Gorm Larsen and Jens Bennedsen

pgl@eng.au.dk and jbb@iha.dk

Department of Engineering

Aarhus School of Engineering

Agenda

- ✓ What did we look at last week
- ➢ Can mathematics add precision to specifications?
- What are Formal Methods?
- Different Formal Methods
- Where are Formal Methods used?
- A guided tour using VDM-SL
- Using the Overture/VDM tool

Mathematical Models

- Abstract representations of a system using mathematical entities and concepts
- Model should captures the essential characteristics of the system while ignoring irrelevant details
- Model can be analyzed using mathematical reasoning to prove system properties or derive new behaviors.
- Two types
  - Continuous models
  - Discrete models (focus here)

Process of Creating a Model

- Clarify requirements and high level design
- Articulate implicit assumptions about environment
- Identify undocumented or unexpected assumptions
- Expose defects
- Identify exceptions

Formal Specification: Why specify

- You may choose to specify because
  - You need additional documentation of your system's interfaces
  - You want a more abstract description of your system design
  - You want to perform some formal analysis of your system

Formal Specification: Why formally specify?

- Depends on what is to be formalized
- What formal methods to use
- What benefits you expect from a formal specification not attainable from just an informal one
- Why not use programming languages to specify?

Agenda

- ✓ What did we look at last week
- ✓ Can mathematics add precision to specifications?
- ➢ What are Formal Methods?
- Different Formal Methods
- Where are Formal Methods used?
- A guided tour using VDM-SL
- Using the Overture/VDM tool

What are Formal Methods?

- **Formal Methods** refers to the use of techniques from logic and discrete mathematics in the specification, design and development of computer systems and software.
- Mastering of **complexity** using **abstraction**.
- Reduce argumentation to a **calculation** which can be checked by mechanical means.
- Replace reviews with a **repeatable** analysis.
- Formal methods can be used at different levels of **rigour**.

Why Consider Formal Methods?

- The development of a formal specification provides insights and an understanding of the software requirements and software design
  - Clarify customers' requirements
  - Reveal and remove ambiguity, inconsistency and incompleteness
  - Facilitate communication of requirement or design
  - Provides a basis for an "elegant" software design
  - Traceability
    - System-level requirements should be traceable to subsystems or components

Advantages using formal specifications

- Specifications must leave as much freedom as possible to the implementor
- Main advantages of formal methods:
    - Abstraction: good mechanism to support implementation freedom
    - Precision: still maintain ability to precisely describe what is needed of the system
- Sometimes it is even possible to refine the formal specification into the final implementation
    - Called formal development process

Validation Techniques

- Inspection: organized process of examining the model alongside domain experts.
- Static Analysis: automatic checks of syntax & type correctness, detect unusual features.
- Testing: run the model and check outcomes against expectations.
- Model Checking: search the state space to find states that violate the properties we are checking.
- Proof: use a logic to reason symbolically about whole classes of states at once.

Different Levels of Rigour

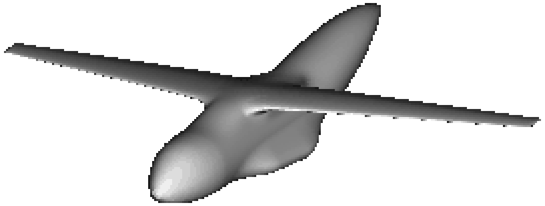
- Level 1: Use of concepts and notation from discrete maths
- Level 2: Use of formalized abstract specification languages with some mechanized support tools
- Level 3: Verification of the abstract precise specification
- Level 4: Fully formal development (refinement from abstract specifications)

FORMAL METHODS SPECIFICATION AND VERIFICATION GUIDEBOOK FOR SOFTWARE AND COMPUTER SYSTEMS

VOLUME I: PLANNING AND TECHNOLOGY INSERTION

J. C. Kelly and K. Kemp, NASA, 1995, page 29

Modelling Computing Systems



Modelling Computing Systems

- In other engineering disciplines (Mechanical, Electrical, Aeronautical etc.) system models are built *to help gain confidence* in requirements and designs.
  - For example: wind tunnels, stress models
- There are two characteristics of these models which are crucial to their successful use: abstraction and rigour.

Abstraction

Engineering models omit details that are not relevant to the purpose of the model. For example:

Windtunnel – assess aerodynamics

Cockpit mockup – assess human factors,
train pilots

The omission of detail not relevant to a model’s purpose is called abstraction. The choice of which details to omit is a matter of engineering skill.

Abstraction

Compare these extracts from two descriptions of the same system.

The FlightFinder System is to be used by travel agents and their customers. Details are entered, including point of departure, destination, preferred dates and times. The system will respond with a range of itineraries and fares, along with the relevant restrictions.

"What"

The system records locations as nodes in a connected graph structure. Each node struct contains an array of pointers to reachable destinations plus, for each pointer, a timetable of flights stored as a hash table. Each record in the hash table has a flight number (8 character string), departure and arrival times (standard time formats) and operating dates (standard date format). To obtain the optimal route, the graph must be traversed using a shortest path algorithm on a modified adjacency matrix ...

"How"

Rigour

The most important property of a model of a computing system is its suitability for analysis: must be

- objective (not down to the opinion of individual engineers)
- repeatable
- susceptible to machine support.

The language in which a model is expressed should be rigorously defined:

- little room for disagreement about what a model actually says
- analysis tools reach the same conclusion about model's properties

Many programming languages have non-rigorous definitions. What is the consequence?

variation – in system behaviour, user understanding

Mathematical Representation of Software

- Formal specifications are expressed in a mathematical notation with precisely defined vocabulary, syntax and semantics.
- Algebraic approach
  - The system is specified in terms of its operations and their relationships.
- Model-based approach
  - The system is specified in terms of a state model that is constructed using mathematical constructs such as sets and sequences.

Where to read about Formal Methods

- Good papers to begin with:
  - “Seven Myths of Formal Methods”, Anthony Hall, IEEE Software 1990.
  - “Formal Methods: State of the Art and Future Directions”, Edmund M. Clarke, Jeannette M. Wing, ACM Computing Surveys, 1996
  - “Ten Commandments of Formal Methods … Ten Years Later”, Jonathan P., Bowen and Mike Hinchey, IEEE Computer, 39(1):40-48, January 2006.
  - “Formal Methods: Practice and Experience”, Jim Woodcock, Peter Gorm Larsen, Juan Bicarregui and John Fitzgerald, ACM Computing Surveys, 41(4), 2009.

Hall: Seven Myths of FM

- Hall (1990) refutes some of these arguments against the use of FM in “seven myths” of FM

1. Perfect software results from the use of FM
2. FM means program Proving
3. FM are very expensive that their use can only be justified in safety and mission critical systems
4. FM require advanced training in math
5. FM increase development costs
6. Clients cannot understand formal specifications
7. FM have only been used for trivial system

Agenda

- ✓ What did we look at last week
- ✓ Can mathematics add precision to specifications?
- ✓ What are Formal Methods?
- ➢ Different Formal Methods
- Where are Formal Methods used?
- A guided tour using VDM-SL
- Using the Overture/VDM tool

Examples of formal methods

- VDM (Vienna Development Method)
- Z
- B
- ASM
- Petri Nets (Coloured PN due to Kurt Jensen)
- Algebraic specification, e.g. CASL
- TLA+
- Alloy
- … around 100 different ones (http://formalmethods.wikia.com/wiki/VL)

Classes of Formal Methods

- Model-based approaches (VDM, Z, B)
- Algebraic approaches (CASL, Act One, Larch, OBJ)
- Transition-based approaches (Statecharts, PROMELA)
- Process algebras (CSP, CCS)
- Logic-based approaches (RTL, TLA)
- Reactive approaches (Petri-nets, SDL)

- Combinations like RAISE (VDM + CSP) and LOTOS (Act One + CCS).
- ISO standards for VDM, Z, LOTOS and ITU standard for SDL

Model-based approaches

- Model-driven
  - Specify admissible system states (or values) at some arbitrary snapshot using mathematical entities such as sets, relation, First Order Predicate Logic (pre-condition/post-conditions, invariants)
  - we get all properties of sets and relations "for free"
    - e.g. Z, VDM and B
- (good) We do not have to spell them out every time we specify a system
- (bad) some of those properties are irrelevant to our system
  - Need to say which properties about the standard mathematical structures are irrelevant

Algebraic approaches

- Specify system behavior in terms of properties (conditional equations) that documents the effect of composing functions
- Need to state explicitly what properties we want our system to have
- Any model that satisfies that theory is deemed to be acceptable
- E.g.:
  - CASL
  - Act One
  - Larch
  - OBJ

Transition-based approaches

- We can characterize admissible system behaviors using the required transitions in state machine
- E.g. for each input state and triggering event, and/or Boolean expression (guard), there exists corresponding output state
- E.g.
  - Statechart
  - PROMELA

Process Algebras

- A process refers to the behavior of a system
- Typically used to describe parallel and distributed systems in an algebraic fashion
- E.g.,
  - Concurrent Sequential Processes (CSP)
  - Calculus of Communicating Systems (CCS)

Logic-based approaches

- Express system behavior as logic expressions over traces of allowed operation calls
- E.g.,
  - Real Time Logic (RTL)
  - Temporal Logic of Actions (TLA)

Reactive approaches

- Specify system behavior completely enough that specification can run on some abstract machine as a structured collection of processes
- E.g.,
  - Petri nets
  - SDL

Agenda

- ✓ What did we look at last week
- ✓ Can mathematics add precision to specifications?
- ✓ What are Formal Methods?
- ✓ Different Formal Methods
- ➢ Where are Formal Methods used?
- A guided tour using VDM-SL
- Using the Overture/VDM tool

Formal Methods in Academia

- Tradition with Abstract Models in Europe
- Focus on Formal Development
- US focus on Automatic Verification
- FM taught at many European Universities
- Spreading from the UK
- Strong push from EU academia for standards of FM
- Recent push for the Grand Challenges

Industrial usage of Formal Methods

- FM are not widely used in industrial setting
    - Software management is inherently conservative and is unwilling to adopt emerging technology
    - Lack of experience and evidence of success
    - Lack of tool support
    - Existing tools are not popular/friendly
    - Steep learning curve
    - Scalability issue
    - Widespread ignorance of current specification techniques and their applicability

Benefits of Formal Specifications

- Higher level of rigor leads to better problem understanding
- Defects are uncovered that would be missed using traditional specification methods
- Allows earlier defect identification
- Formal specification language semantics allow checks for self-consistency
- Enables the use of formal proofs to establish fundamental system properties and invariants

Limitations to Formal Methods

- Requires a sound mathematical knowledge of the developer
- Different aspects of a design may be represented by different formal specification methods
- Useful for consistency checks, but formal methods cannot guarantee the completeness of a specifications
- For the majority of systems it does not offer significant cost or quality advantages over others

Tool support for Specification language

- Tool support for modeling include
  - Editors (visual/textual)
- Tool support for analysis
  - Syntactic checking
  - Type checking
  - Interpretation
  - Model checking
  - Theorem Proving
- Synthesis
  - Refinement
  - Code generation
  - Test case generation

Formal Methods Tool Support

- VDMTools from CSK Systems
- Atelier-B from ClearSy
- FDR from Formal Systems Europe
- SCADE from Esterel Technologies
- Lots of prototype/academic tools
- Overture open-source development for VDM

Example Industrial Projects

- DDC Ada Compiler
- CICS
- Sizewell B
- Transputer
- CDIS
- SACEM
- Météor
- SHOLIS
- Mondex smart card
- Storm Surge Barrier Control System

- ConForm
- DustExpert
- CAVA
- Dutch DoD
- BPS 1000
- Flower Auction
- SPOT4
- TradeOne
- FelicaNetworks

DDC Ada Compiler (197X)

- Organisation: DDC (Denmark)
- Domain: Compiler
- Tools: VDM notation
- Experience:
  - First European validated Ada compiler
  - Cheaper than without FM
  - No verification or validation
  - Tools are lacking

CICS (198X)

- Organisation: IBM (UK)
- Domain: Transaction Processing
- Tools: Z notation
- Experience:
  - Reduction in development cost: 9%
  - Code developed from Z had 2½ times fewer problems
  - Parsing, type checking tools increased productivity

Sizewell B (198X)

- Organisation: TACS, UK
- Domain: Nuclear
- Tools: Malpas
- Experience:
    - 100000 lines of code
    - Formal verification at source code
    - About 200 man years of effort!
    - Still favorable compared to retesting

Transputer

- Organisation: INMOS, UK
- Domain: Processor with floating-point hardware
- Tools: Z notation, HOL, CCS
- Experience:
  - IEEE standard formalised in Z
  - occam code derived from Z
  - Many algorithm errors discovered
  - Commercial success
  - Queen’s Award for Technological Achievements

CDIS (199X)

- Organisation: Praxis, UK
- Domain: Air Traffic Control
- Tools: VDM and CCS notations
- Experience:
  - No net cost in using formal methods
  - Quality of the software was much higher
  - Defect rate of about 0.75 faults per KLOC
  - Tools from VIP too prototypic
  - Praxis currently involved in new generation ATC

SACEM (198X)

- Organisation: GEC Alstom now Siemens, France
- Domain: Railways (Paris RER)
- Tools: B-Toolkit (early version)
- Experience:
  - Formal refinement in B
  - Verification but no animation
  - Same price as for non FM development
  - Tools improved during project
  - High customer satisfaction

Météor (199X)

- Organisation: Matra Transport now Siemens, France
- Domain: Railways (Paris Metro)
- Tools: Atelier-B
- Experience:
  - Cost savings for safety-critical software
  - Formal refinements
  - Formal verification
  - Tools improved during project

SHOLIS (199X)

- Organisation: Praxis (UK)
- Domain: Avionics (Military, Helicopter)
- Tools: SPARK and CADiZ
- Experience:
  - Z Specification verification efficient
  - SPARK code verification less efficient
  - Better tool support needed
  - Also used for Lockheed’s C130J

Mondex Smart Card (1999)

- Organisation: LOGICA, UK
- Domain: Smart card for e-finance
- Tools: Z notation
- Experience:
    - Big security risk of financial loss
    - Security verified by hand
    - Major security flaw detected
    - First successful ITSEC E6
    - Repeated on other projects
    - Later redone in connection with the Grand Challenges

Storm Surge Barrier Control System (1996-97)

- Organisation: CMG (now LOGICA), The Netherlands
- Domain: Critical application against flooding
- Tools: Z and PROMELA
- Experience:
  - The use of formal methods has helped the project achieve the required software quality goals
  - The learning curve for formal methods is steep
  - The combination of formal methods and code reviews/module testing pays off for analyzing subtle behavior

ConForm (1994)

- Organisation: British Aerospace (UK)
- Domain: Security (gateway)
- Tools: The VDM-SL Toolbox
- Experience:
  - Prevented propagation of error
  - Successful technology transfer
  - At least 4 more applications without support
- Statements:
  - “Engineers can learn the technique in one week”
  - “**VDMTools**® can be integrated gradually into a traditional existing development process”

DustExpert (1995-7)

- Organisation: Adelard (UK)
- Domain: Safety (dust explosives)
- Tools: The VDM-SL Toolbox
- Experience:
  - Delivered on time at expected cost
  - Large VDM-SL specification
  - Testing support valuable
- Statement:
  - “Using **VDMTools**® we have achieved a productivity and fault density far better than industry norms for safety related systems”

Adelard Metrics

Initial requirements	450 pages
VDM specification	16kloc (31 modules) 12kloc (excl comments)
Prolog implementation	37kloc 16kloc (excl comments)
C++ GUI implementation	23kloc 18kloc (excl comments)

- 31 faults in Prolog and C++ (< 1/kloc)
- Most minor, only 1 safety-related
- 1 (small) design error, rest in coding

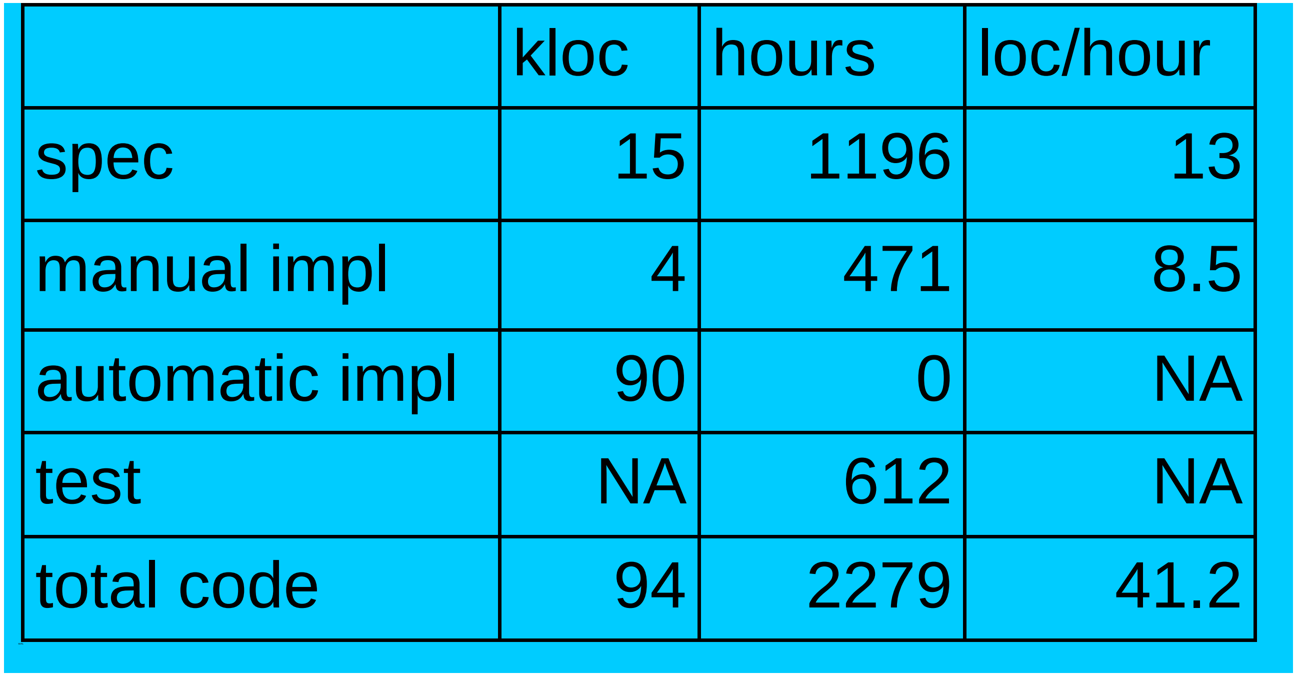
CAVA (1998-)

- Organisation: Baan (Denmark)
- Domain: Constraint solver (Sales Configuration)
- Tools: The VDM-SL Toolbox
- Experience:
  - Common understanding
  - Faster route to prototype
  - Earlier testing
- Statement:
  - "**VDMTools**® has been used in order to increase quality and reduce development risks on high complexity products"

Dutch DoD (1997-8)

- Organisation: Origin, The Netherlands
- Domain: Military
- Tools: The VDM-SL Toolbox
- Experience:
  - Higher level of assurance
  - Mastering of complexity
  - Delivered at ***expected cost*** and ***on schedule***
  - ***No errors detected in code after delivery***
- Statement:
  - "We chose **VDMTools**® because of high demands on maintainability, adaptability and reliability"

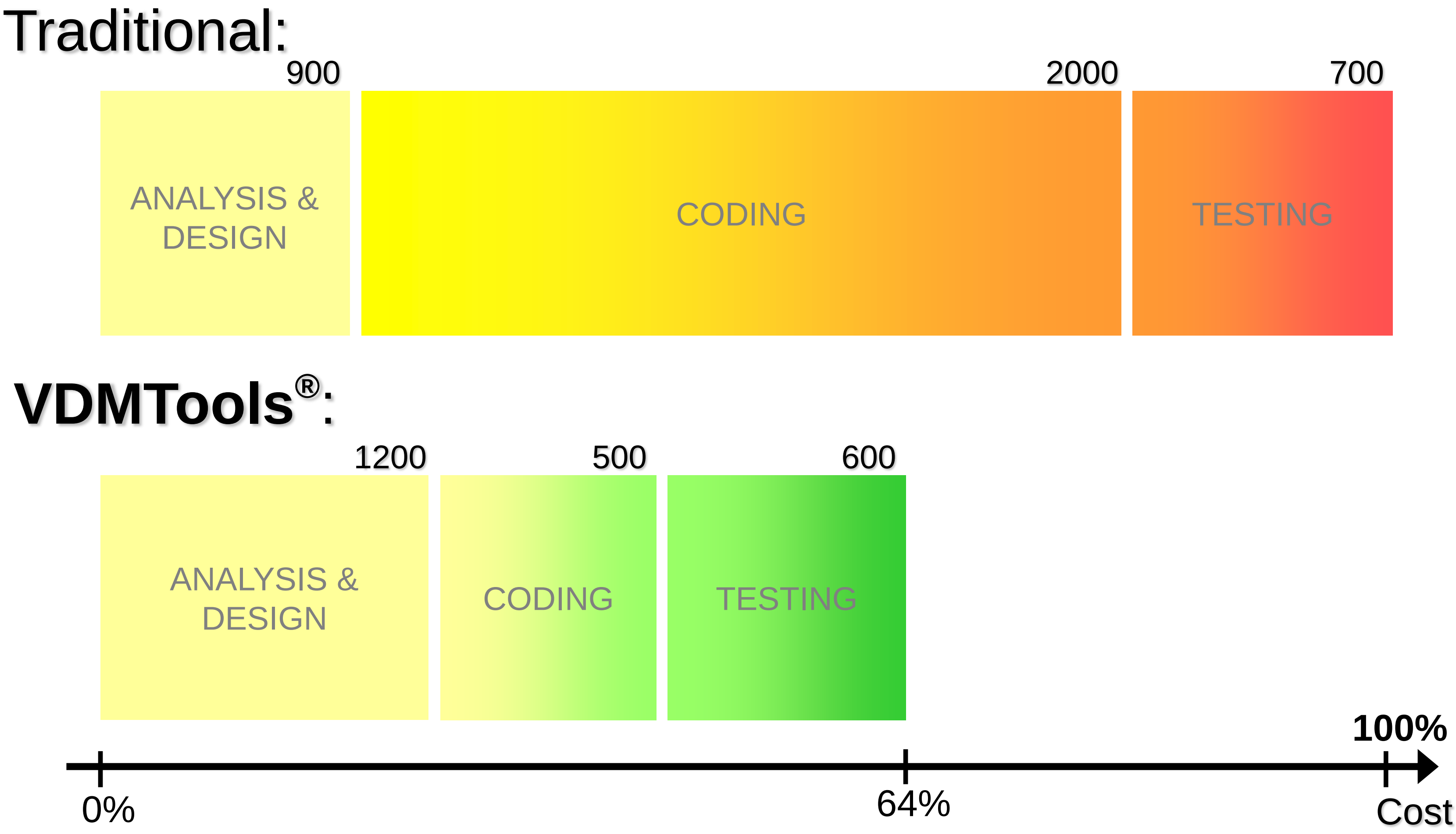
DoD, NL Metrics (1)



	kloc	hours	loc/hour
spec	15	1196	13
manual impl	4	471	8.5
automatic impl	90	0	NA
test	NA	612	NA
total code	94	2279	41.2

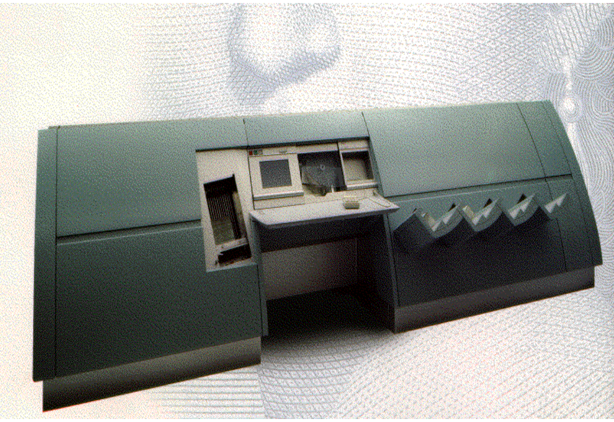
- Estimated 12 C++ loc/h with manual coding!

DoD - Comparative Metrics



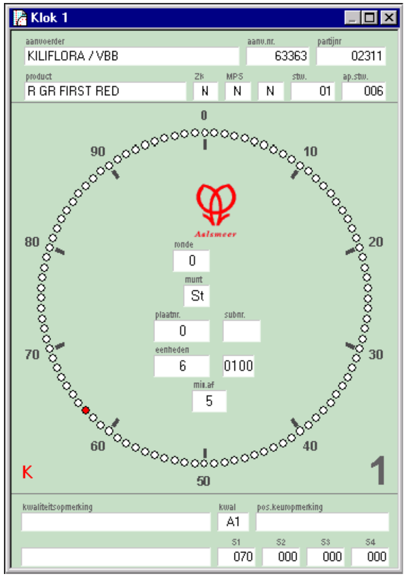
BPS 1000 (1997-)

- Organisation: GAO, Germany
- Domain: Bank note processing
- Tools: The VDM-SL Toolbox
- Experience:
  - Better understanding of sensor data
  - Errors identified in other code
  - Savings on maintenance
- Statement:
  - VDMTools provides unparalleled support for design abstraction ensuring quality and control throughout the development life cycle.



Flower Auction (1998)

- Organisation: Chess, The Netherlands
- Domain: Financial transactions
- Tools: The VDM++ Toolbox
- Experience:
  - Successful combination of UML and VDM++
  - Use iterative process to gain client commitment
  - Implementers did not even have a VDM course
- Statement:
  - “The link between VDMTools and Rational Rose is essential for understanding the UML diagrams”

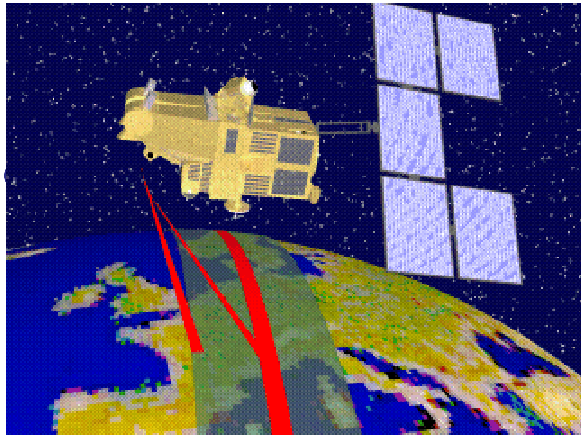


aanvoerder: KILIFLORA / VBB
aanv.nr.: 63363
partijnr: 02311
product: R GR FIRST RED
Zk: N
MPS: N
N
stw.: 01
ap.stw.: 006
ronde: 0
munt: St
plaatnr.: 0
subnr.:
eenheden: 6
0100
min.af: 5
kwaliteitsopmerking:
kwal: A1
pos.keuropmerking:
S1: 070
S2: 000
S3: 000
S4: 000

SPOT 4 (1999)

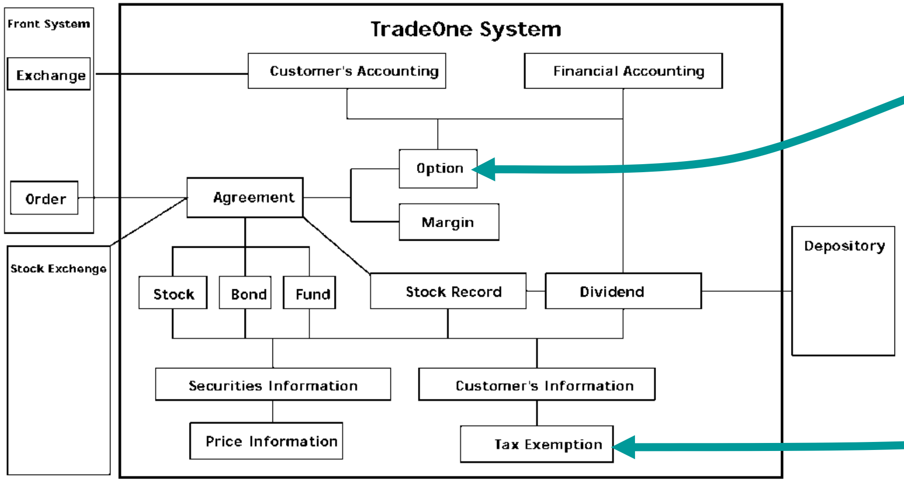
- Organisation: CS-CI, France
- Domain: Space (payload for SPOT4 satellite)
- Tools: The VDM-SL Toolbox
- Experience:
  - 38 % less lines of source code
  - 36 % less overall effort
  - Use of automatic C++ code generation
- Statement:

  The cost of applying Formal methods is si
  lower than without them.



TradeOne, CSK, 2000 - 2001

- Full TradeOne system is 1.3 MLOC system
- Mission-critical backbone system keeping track of financial transactions conducted
- Used by securities companies and brokerage houses



Options Subsystem handles the business process for trading options. Modelled in VDM++

Tax exemption subsystem has particularly complex regulations to implement. Modelled in VDM++.

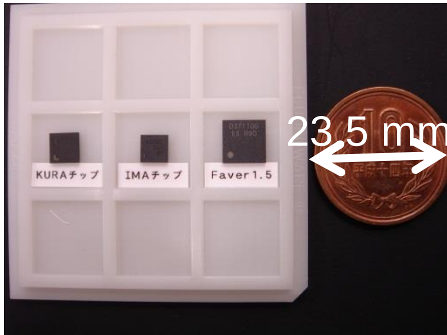
TradeOne Cost Effectiveness

Subsystem	COCOMO estimate	Real time	Time saving
Tax exemption	Effort:38.5 PM Schedule:9M	Effort:14 PM Schedule: 3.5 M	Effort:74% Schedule:61%
Options	Effort:147.2 PM Schedule:14.3M	Effort: 60.1 PM Schedule:7M	Effort: 60% Schedule: 51%

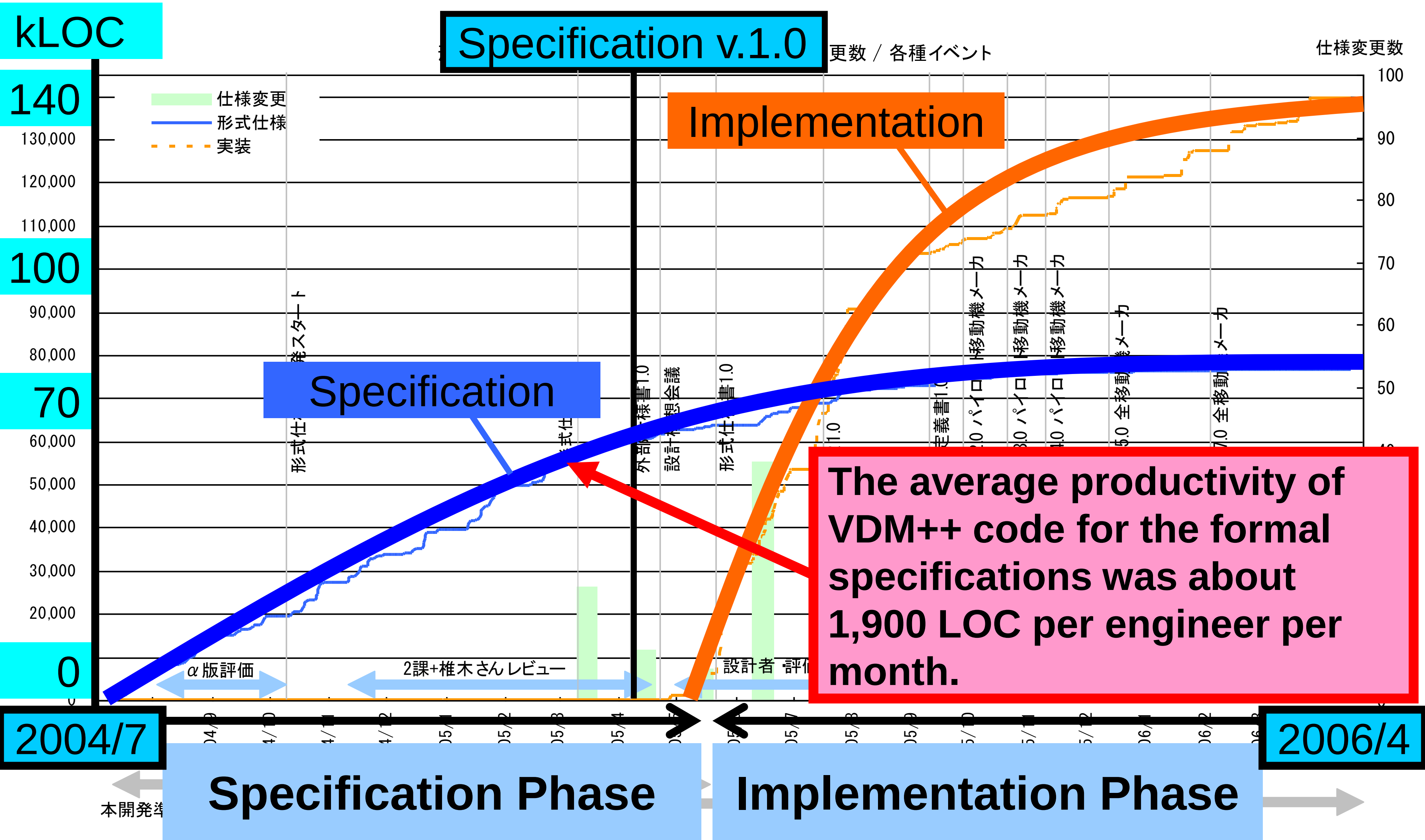
Overall sizes	
Total TradeOne	1,342,858
Tax exemption subsystem	18,431
Option subsystem	60,206

The FeliCa Mobile Chip Project

- Mobile FeliCa IC chips can be embedded inside mobile phones
- Used for different on-line services including payment
- Uses Near-Field-Communication technology
- Used for example for metro ticking in Tokyo
- The IC Chips contains an operating system as firmware for more than 125 million mobile phones
- This is fully developed using the VDM++ technology
- Between 50 and 60 people in total on the project

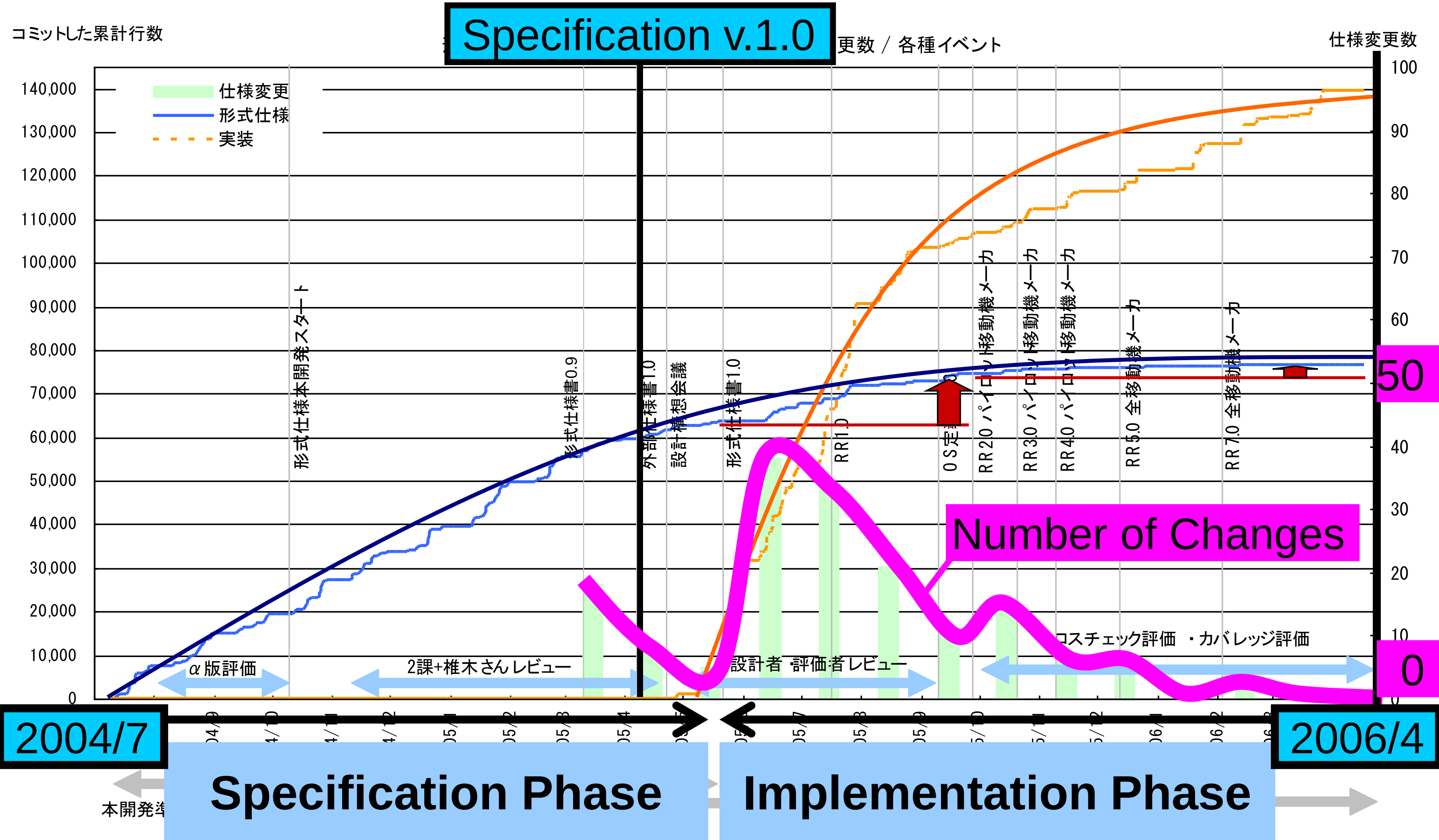


Specification and Implementation Growth



The average productivity of VDM++ code for the formal specifications was about 1,900 LOC per engineer per month.

Number of Changes



Specification v.1.0

50

Number of Changes

0

2004/7

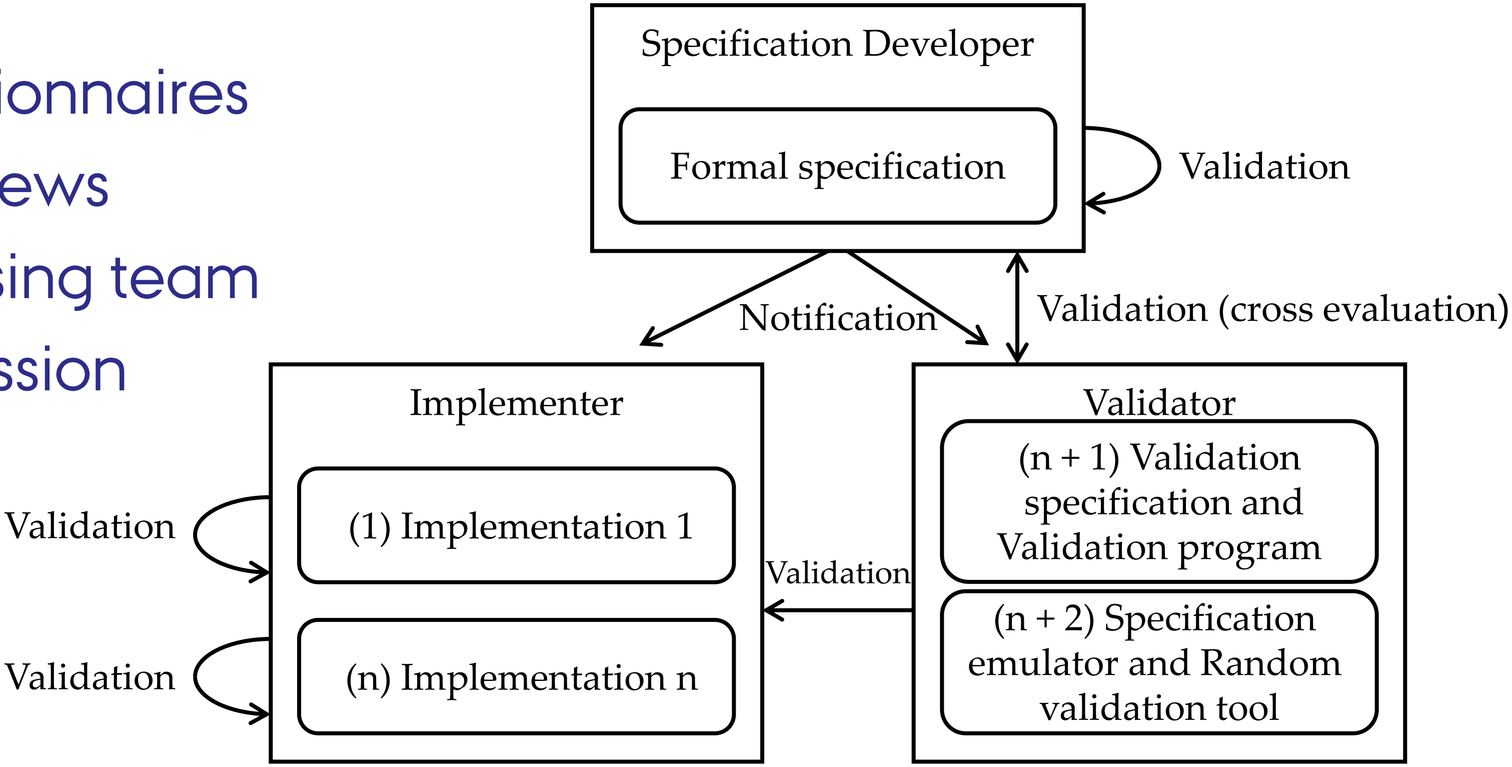
2006/4

Specification Phase

Implementation Phase

Development Team Structure

- Divided into specification, implementation and validation teams
- VDM++ used for improving communication between teams
- Questionnaires
- Interviews
- Assessing team impression



Comparing the projects

- **CSK Systems**
  - Requirements specification written with use cases was modeled using VDM++ (VDM++ modelling team: 2 – 5 people).
  - Defect density: 6.25% of the product norm (zero defect since release).
  - Productivity: 2.5 times COCOMO prediction.
  - Development period: 45% of COCOMO. Finished on schedule.
- **FeliCa Networks**
  - Operating system of Felica chip firmware modeled using VDM++ (Specification description team: ~20 people).
  - Zero defect since release (discovered ~5 times more defects using VDM++ than with conventional review).
  - Development period ~3 yrs. Finished on schedule.

Specification Description

	CSK	FeliCa Networks
Japanese specification	Equivalent to Use cases	Combination of UML and VDM
VDM Model	Make use cases rigorous	File Specification Command Spec.
Lines of VDM model	11,757 + 68,170 = Approximately 80,000	Approximately 100,000
Lines of testing framework and test cases in VDM	Approximately 50,000	Approximately 60,000
Lines of implementation	18,431 + 64,377 = Approximately 80,000	Approximately 110,000
Framework	Hierarchical structure + validation using testing	Hierarchical structure + validation using testing
Library	General purpose + Business	Business
VDM written in Japanese	Yes	No

Agenda

- ✓ What did we look at last week
- ✓ Can mathematics add precision to specifications?
- ✓ What are Formal Methods?
- ✓ Different Formal Methods
- ✓ Where are Formal Methods used?
- ➢ A guided tour using VDM-SL
- • Using the Overture/VDM tool

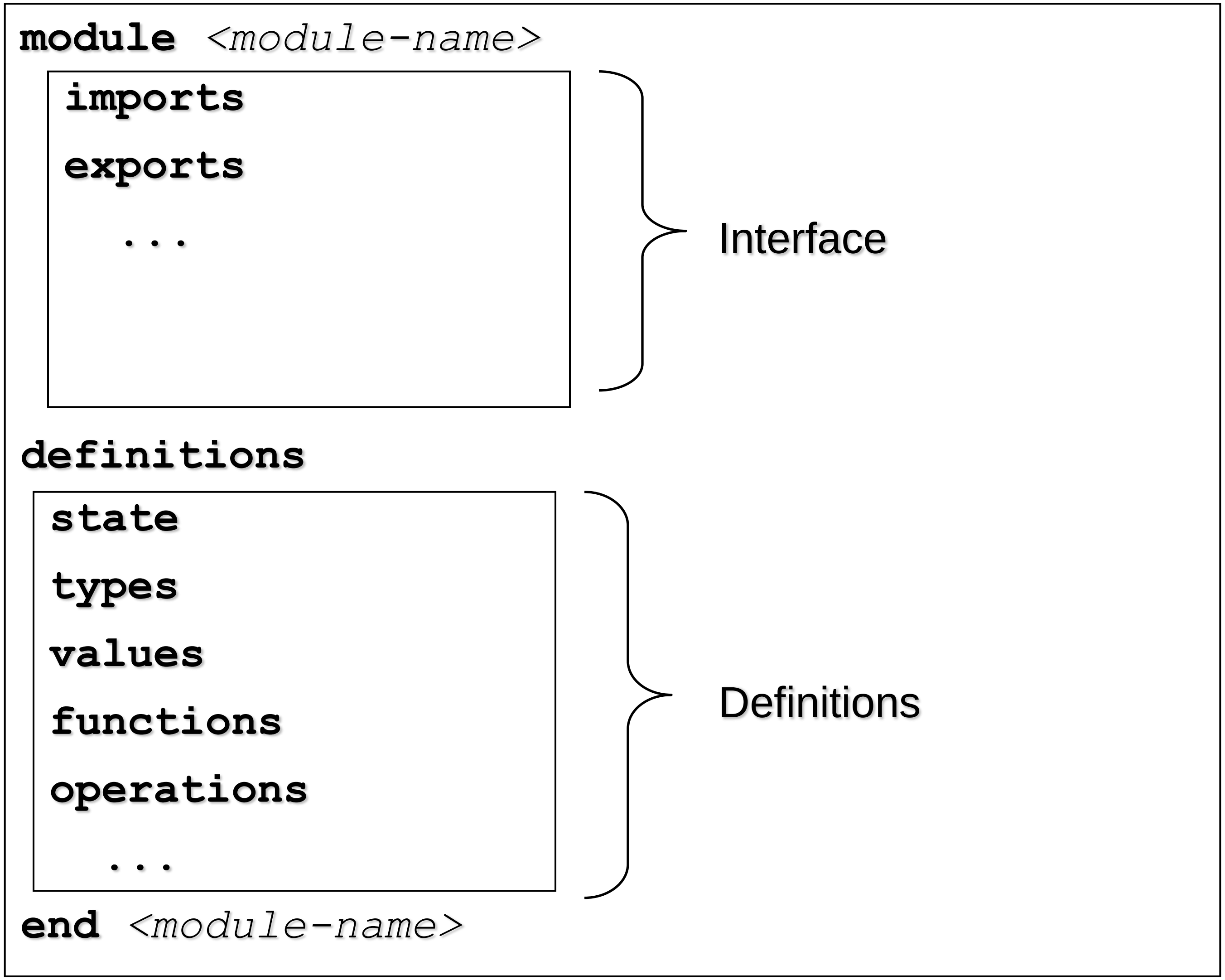
Before we start

- Overview of VDM
- Writing complete VDM spec is not the goal
- Some of this should be familiar
  - Did you take Discrete Math?
- Syntax can be scary
- **Ask questions!**

The Vienna Development Method

- Invented at IBM's labs in Vienna in the 70's
- 3 Dialects: SL, ++ and RT
- Industry strength tools are freely available
  - VDMTools: http://www.vdmtools.jp/en
  - Overture open source tool:
    https://github.com/overturetool/overture
- Model-oriented specification:
  - Types specify data
  - Functions specify behavior

VDM-SL Module Outline



```
module <module-name>
  imports
  exports
     ...

definitions
  state
  types
  values
  functions
  operations
     ...
end <module-name>
```

Interface

Definitions

The contents of a model in VDM-SL

- **Data Types**
  - basic types: **`int`**, **`char`**, **`bool`**…
  - new types built from existing ones: sets, mappings, records
  - Invariants restrict the values of a type
- **Functions** define behavior
  - Logic, arithmetic, etc.
  - Pure
  - Pre-conditions restrict input values

The contents of a model in VDM-SL

Type Definitions, e.g.

```
Altitude = real
inv alt == alt >= 0

Position :: lat  : Latitude
            long : Longitude
            alt  : Altitude
```

Function definitions, e.g.

```
Move: Id * Position * ATCSystem  ->  ATCSystem
Move(id, pos, sys) == … expression …
pre … expression …
```

The contents of a model in VDM-SL

- Data abstraction
  - Unconstrained data types
  - Collections are unbounded
- Function abstraction
  - Implicit specification
  - Pre and post conditions relate input and output
  - No algorithm provided

```
SquareRoot(x:nat)r:real
pre  x >= 0
post r*r = x
```

Building Formal Model

- Many possible approaches
- Always consider the model’s **purpose**!

1. *Read the requirements.*
2. *Extract a list of possible data types (often from nouns) and functions (often from verbs/actions).*
3. *Sketch out representations for the data types.*
4. *Sketch out signatures for the functions.*
5. *Complete type definitions by determining invariants.*
6. *Complete the function definitions, modifying data type definitions if required.*
7. *Review the requirements, noting how each clause has been treated in the model.*

Constructing a Model

The Chemical Plant Alarm System

Example derived from an alarm and callout system developed by IFAD, a Danish high-technology firm for the local telephone company Tele Danmark Process.

Requirements for the Alarm Example

A chemical plant has monitors which can raise alarms in response to conditions in the plant. When an alarm is raised, an expert must be called to the scene. Experts have different qualifications for coping with different kinds of alarm.

Requirements for the Alarm Example

***R1**: A computer-based system is to be developed to manage expert call-out in response to alarms.*

***R2**: Four qualifications: electrical, mechanical, biological and chemical.*

***R3**: There must be experts on duty at all times.*

***R4**: Each expert can have a whole list of qualifications, not just one.*

***R5**: Each alarm has a description (text for the expert) and a qualification.*

***R6**: When an alarm is raised, the system should output the name of a qualified and available expert who can then be called in.*

***R7**: It shall be possible to check when a given expert is available.*

***R8**: It shall be possible to assess the number of experts on duty at a given period*

Purpose of the model

Clarify the rules governing duty roster and calling of experts to respond to alarms

In professional practice, the purpose of the model is seldom made clear…

Possible Types and Functions

Types

Plant

Qualification

Alarm

Period

Expert

Description

Functions

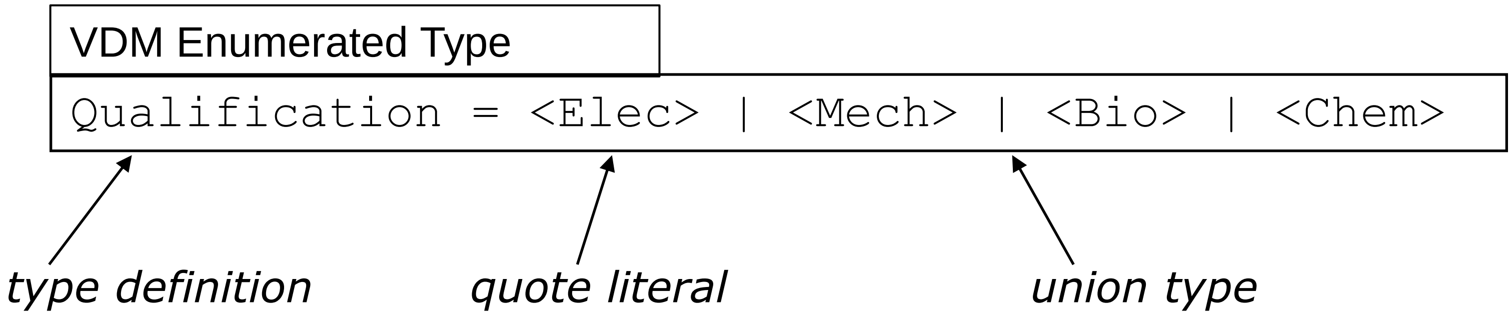
ExpertToPage

ExpertIsOnDuty

NumberOfExperts

Sketch Types

R2: Four qualifications: electrical, mechanical, biological and chemical.



VDM Enumerated Type

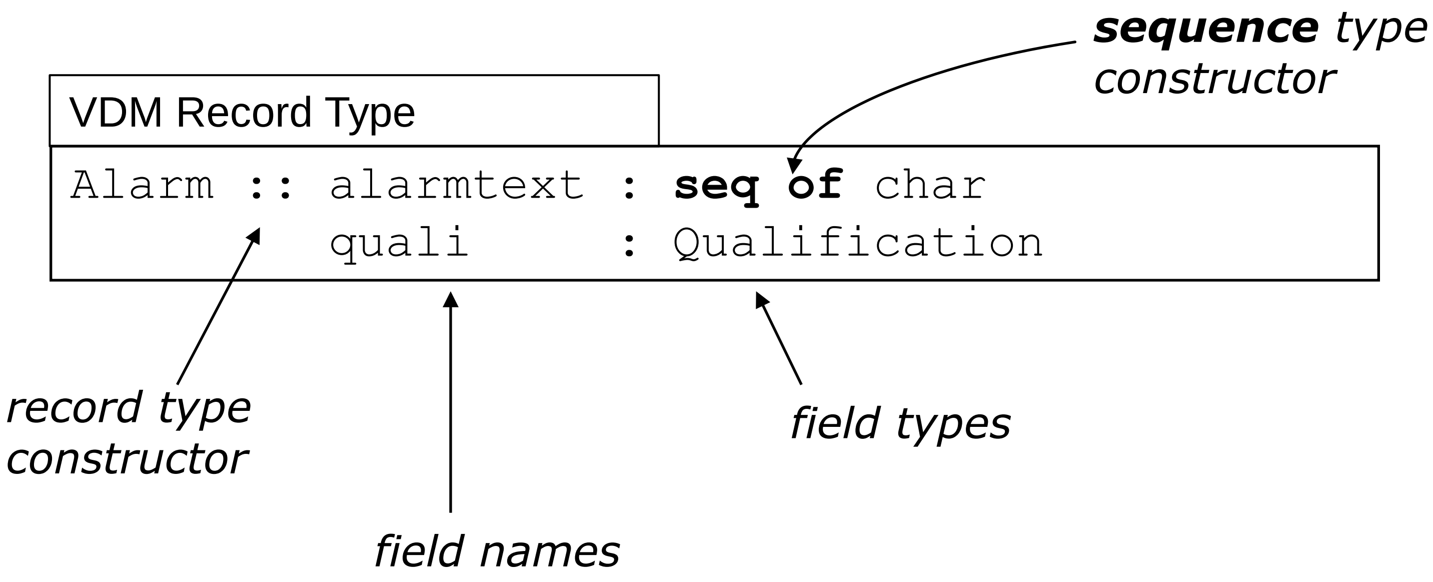
```
Qualification = <Elec> | <Mech> | <Bio> | <Chem>
```

- Similar to Enums from Java, C#, etc.
- | constructs union of types
- Individual value in <>
- This type maps to the four qualifications

Sketch Types

R5: Each alarm has a description (text for the expert) and a qualification.

What is "a"? Some? At least one?



VDM Record Type

```
Alarm :: alarmtext : seq of char
         quali     : Qualification
```

record type constructor

field names

field types

***sequence** type constructor*

Sketch Types

```
Alarm :: alarmtext : seq of char
         quali     : Qualification
```

v : T	v has type T
a : Alarm	a is an alarm
a.alarmtext	field access
a = mk_Alarm(“Alert”, <Elec>)	mk_ is the record constructor

Sketch Types

R4: Each expert can have a whole list of qualifications, not just one.

Lists have order. Does that matter?

```
Expert :: expertId : ExpertId
          quali    : set of Qualification
```

- Natural language requirements often imprecise
- When in doubt, clarify
- Model as abstract as possible – only record relevant details
- Requires skill, experience

Sketch Types

- Requirements give no indication about expert identifier
  - Same for timetable period
- Need a type but no detailed representation
- Use **token** – simplest type

VDM Token Type

```
ExpertId = token

Period = token
```

Sketch Types

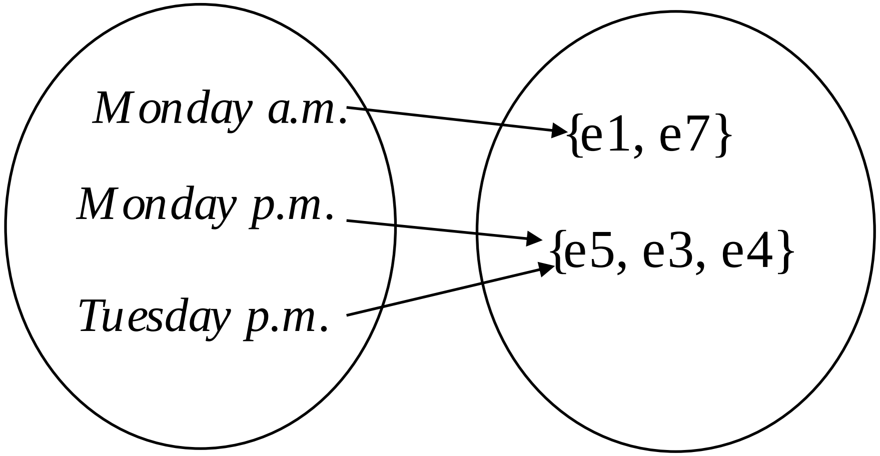
R3: There must be experts on duty at all times.

R7: It shall be possible to check when a given expert is available.

- Relation between time period and expert on duty

VDM Mapping Type

```
Schedule = map Period to (set of Expert)
```



Sketch Types

R1: A computer-based system is to be developed to manage expert call-out in response to alarms.

```
Plant :: sch    : Schedule
         alarms : set of Alarm
```

Whole plant

Model so far - types

```
Plant :: sch      : Schedule
         alarms : set of Alarm

Schedule = map Period to set of Expert

Period = token

Expert :: expertid : ExpertId
          quali    : set of Qualification

ExpertId = token

Qualification = <Elec> | <Mech> | <Bio> | <Chem>

Alarm :: alarmtext : seq of char
         quali     : Qualification
```

Sketching function signatures

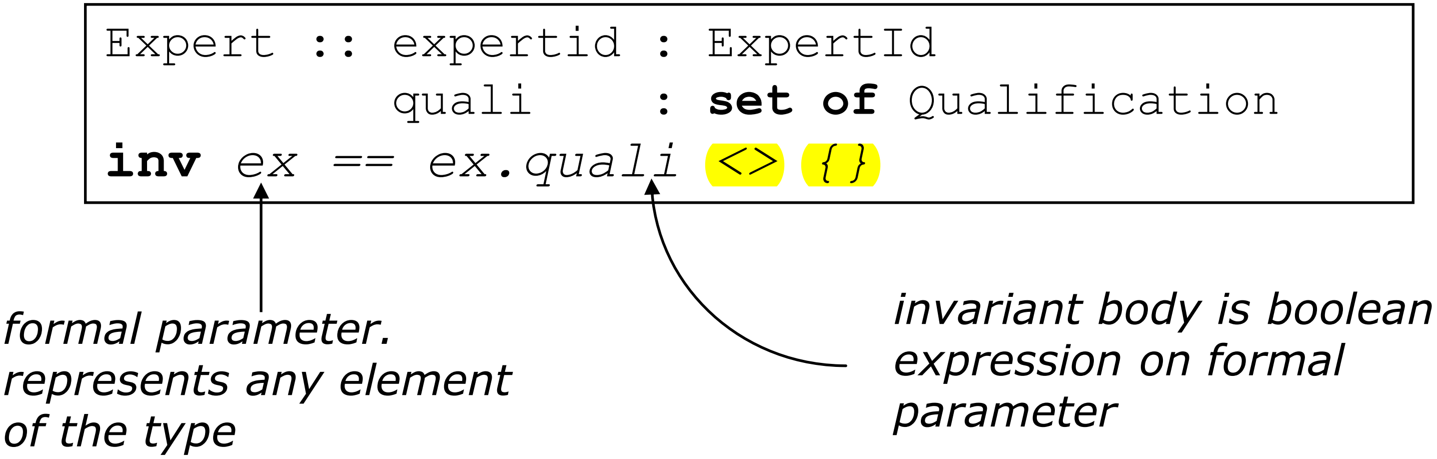
- Possible functions:
  - `ExpertToPage`
  - `ExpertIsOnDuty`
  - `NumberOfExperts`
- Function signatures show input and result types

Function Signatures

```
ExpertToPage: Alarm * Period * Plant -> Expert
ExpertIsOnDuty: Expert * plant -> set of Period
NumberOfExperts: Period * Plant -> nat
```

Complete type definitions

- Restrictions on values of the system
- Must hold at all times
- Type invariants



```
Expert :: expertid : ExpertId
          quali    : set of Qualification
inv ex == ex.quali <> { }
```

formal parameter. represents any element of the type

invariant body is boolean expression on formal parameter

What does this invariant mean?

Complete type definitions

R3: There must be experts on duty at all times.

- Restriction on schedule
- For all periods, set of experts cannot be empty
- How do we write this invariant?

```
Schedule = map Period to set of Expert
inv sch == forall exs in set rng sch & exs <> {}
```

Complete function definitions

- Function definition contains:
  - Signature (already sketched)

    ```
    NumberOfExperts: Period * Plant -> nat
    ```

  - Parameters

    ```
    NumberOfExperts(peri,pl) ==
    ```

  - Body

    ```
    card pl.sch(peri)
    ```

  - Pre-condition (optional)

    ```
    pre peri in set dom pl.sch
    ```

Complete function definitions

R7: It shall be possible to check when a given expert is available.

```
ExpertIsOnDuty: Expert * Plant -> set of Period
ExpertIsOnDuty(ex,pl) ==
    {peri | peri in set dom pl.sch &
            ex in set pl.sch(peri)}
```

Complete function definitions

R7: It shall be possible to check when a given expert is available.

- Alternatively, pattern-match on input
- Unpack record at parameter for direct access to fields

```
ExpertIsOnDuty: Expert * Plant -> set of Period
ExpertIsOnDuty(ex,mk_Plant(sch,alarms)) ==
    {peri | peri in set dom sch &
            ex in set sch(peri)}
```

Complete function definitions

R7: It shall be possible to check when a given expert is available.

- `alarms` not used in function
- replace with `-` pattern ("don't care")

```
ExpertIsOnDuty: Expert * Plant -> set of Period
ExpertIsOnDuty(ex, mk_Plant(sch,-)) ==
   {peri | peri in set dom sch & ex in set sch(peri)}
```

Complete function definitions

R6: When an alarm is raised, the system should output the name of a qualified and available expert who can then be called in.

```
ExpertToPage: Alarm * Period * Plant -> Expert
ExpertToPage(al,peri,pl) == ???
```

- How to choose an expert is not relevant at this time
- Specify result without specifying the "how"
- Use implicit function

```
ExpertToPage(al:Alarm, peri:Period, pl:Plant) r:Expert
pre   …
post  r in set pl.sch(per) and
      al.quali in set r.quali
```

implicit functions have a different header and no signature

Any problems with the system?

- No mention of qualification availability
- After consulting client, it is necessary to have one expert with each qualification available at all times

```
Plant :: sch    : Schedule
         alarms : set of Alarm
inv mk_Plant(sch,alarms) ==
    forall a in set alarms &
        forall per in set dom sch &
            exists ex in set sch(per) &
                a.quali in set ex.quali
```

Review requirements

***R1**: system to manage expert call-out in response to alarms.*

***R2**: Four qualifications.*

***R3**: experts on duty at all times.*

***R4**: expert can have list of qualifications.*

***R5**: Each alarm has description & qualification.*

***R6**: output the name of a qualified and available expert*

***R7**: check when a given expert is available.*

***R8**: assess the number of experts on duty at a given period*

Problems with requirements

- At least one suitable expert must be available
- Implicit expert identifiers
- “List” meant “set”
- Experts without qualifications are useless
- “A qualification” meant “several qualifications”.

Agenda

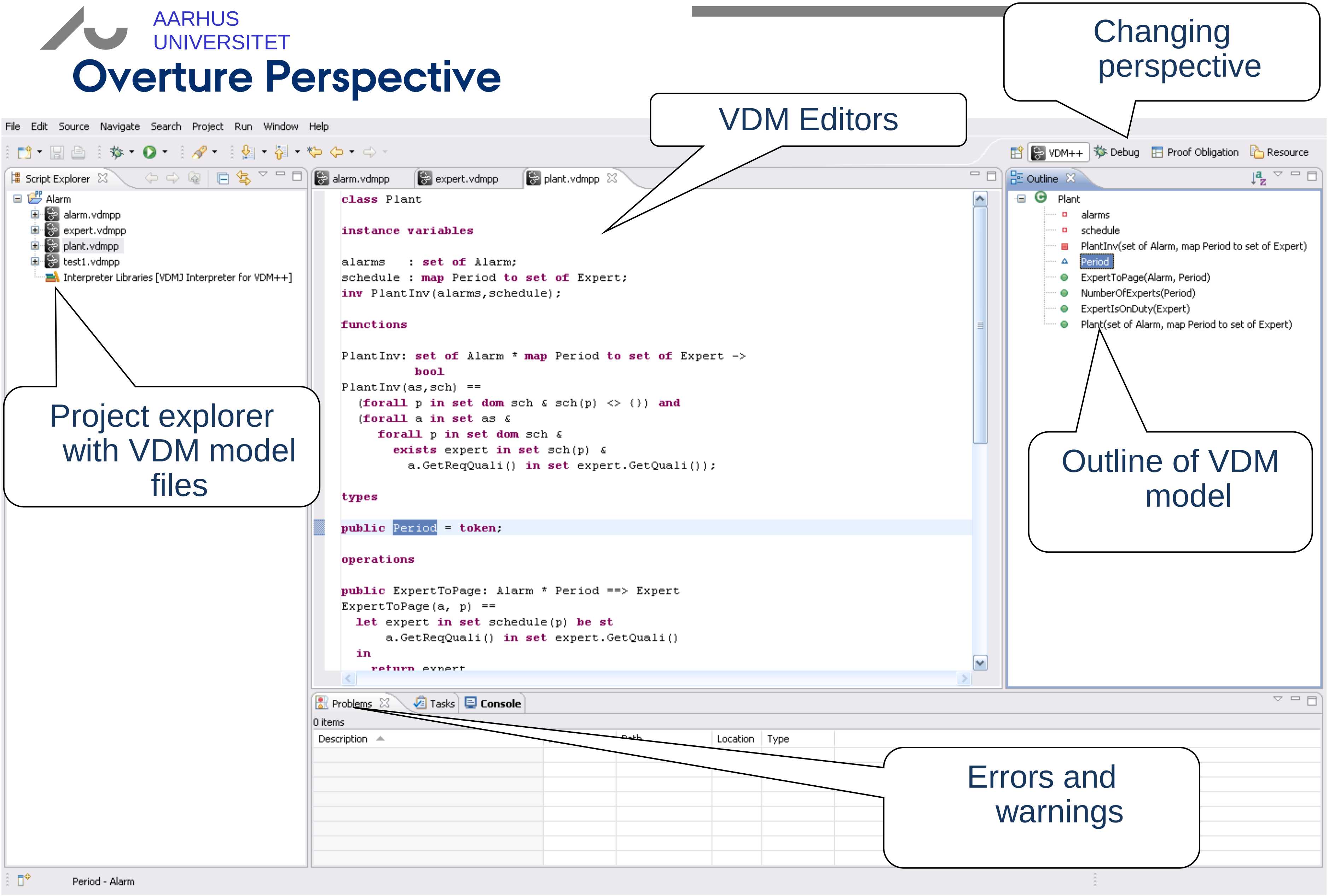
- ✓ What did we look at last week
- ✓ Can mathematics add precision to specifications?
- ✓ What are Formal Methods?
- ✓ Different Formal Methods
- ✓ Where are Formal Methods used?
- ✓ A guided tour using VDM-SL
- ➢ Using the Overture/VDM tool

The Overture/VDM tool

- Open source
- Built on Java/Eclipse
- Plug-in architecture
- Tool available from:
  - https://sourceforge.net/projects/overture/ (use 2.0)
- Documentation available from:
  - http://wiki.overturetool.org/index.php/Overture_Publications
- Select VDM-SL project, include selected VDM-SL file
- Write VDM inside blocks:
  - `\begin{vdm_al}`
  - `\end{vdm_al}`
- Write explanations outside blocks

AARHUS
UNIVERSITET

Overture Perspective



```
class Plant

instance variables

alarms   : set of Alarm;
schedule : map Period to set of Expert;
inv PlantInv(alarms,schedule);

functions

PlantInv: set of Alarm * map Period to set of Expert ->
          bool
PlantInv(as,sch) ==
  (forall p in set dom sch & sch(p) <> {}) and
  (forall a in set as &
     forall p in set dom sch &
       exists expert in set sch(p) &
         a.GetReqQuali() in set expert.GetQuali());

types

public Period = token;

operations

public ExpertToPage: Alarm * Period ==> Expert
ExpertToPage(a, p) ==
  let expert in set schedule(p) be st
      a.GetReqQuali() in set expert.GetQuali()
  in
    return expert
```

Changing perspective

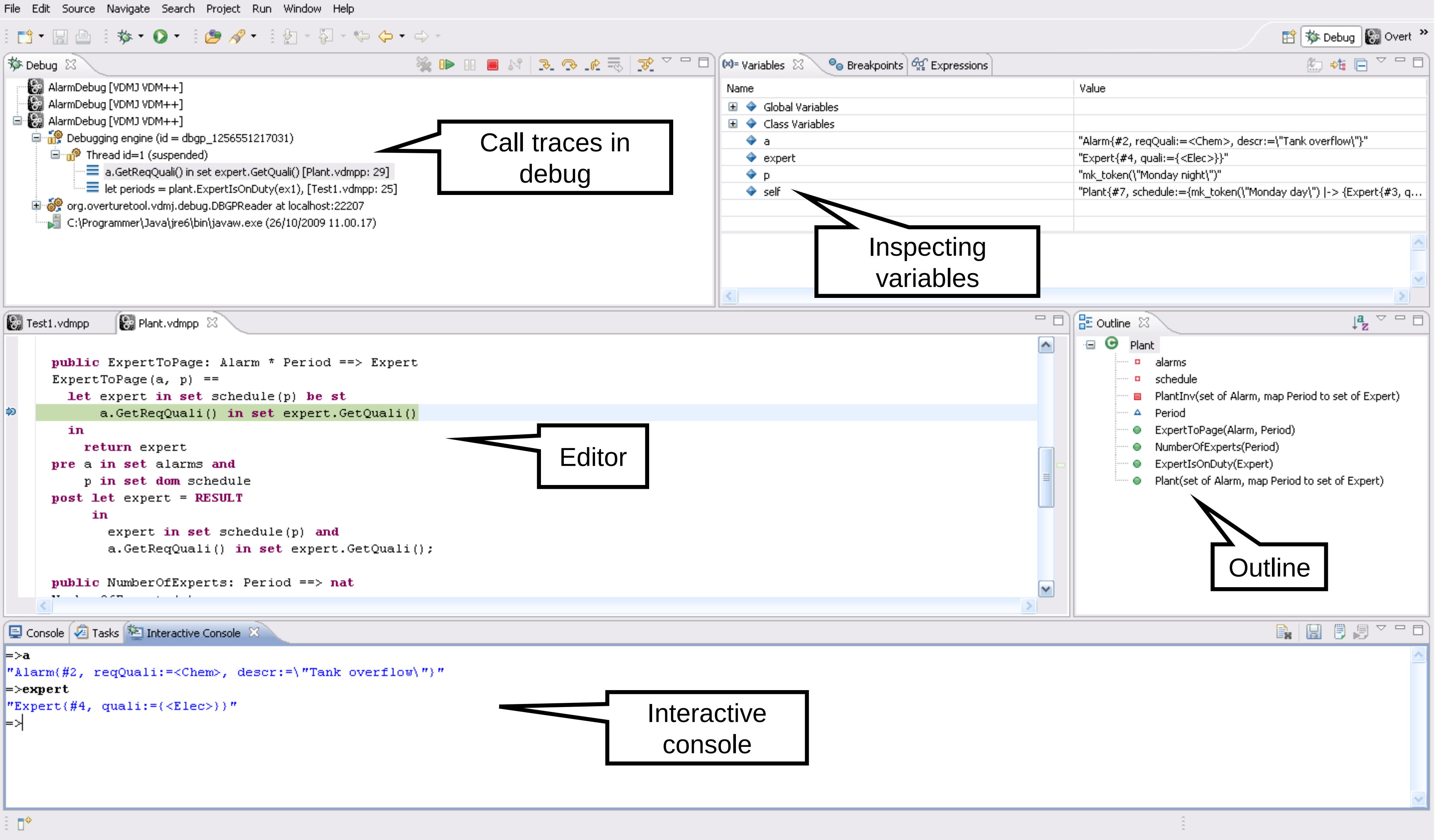
VDM Editors

Project explorer with VDM model files

Outline of VDM model

Errors and warnings

Debug Perspective



```
AlarmDebug [VDMJ VDM++]
AlarmDebug [VDMJ VDM++]
AlarmDebug [VDMJ VDM++]
    Debugging engine (id = dbgp_1256551217031)
        Thread id=1 (suspended)
            a.GetReqQuali() in set expert.GetQuali() [Plant.vdmpp: 29]
            let periods = plant.ExpertIsOnDuty(ex1), [Test1.vdmpp: 25]
    org.overturetool.vdmj.debug.DBGPReader at localhost:22207
    C:\Programmer\Java\jre6\bin\javaw.exe (26/10/2009 11.00.17)
```

Call traces in debug

Name	Value
Global Variables	
Class Variables	
a	"Alarm{#2, reqQuali:=<Chem>, descr:=\"Tank overflow\"}"
expert	"Expert{#4, quali:={<Elec>}}"
p	"mk_token(\"Monday night\")"
self	"Plant{#7, schedule:={mk_token(\"Monday day\") |-> {Expert{#3, q...

Inspecting variables

```
public ExpertToPage: Alarm * Period ==> Expert
ExpertToPage(a, p) ==
  let expert in set schedule(p) be st
      a.GetReqQuali() in set expert.GetQuali()
  in
    return expert
pre a in set alarms and
    p in set dom schedule
post let expert = RESULT
     in
       expert in set schedule(p) and
       a.GetReqQuali() in set expert.GetQuali();

public NumberOfExperts: Period ==> nat
```

Editor

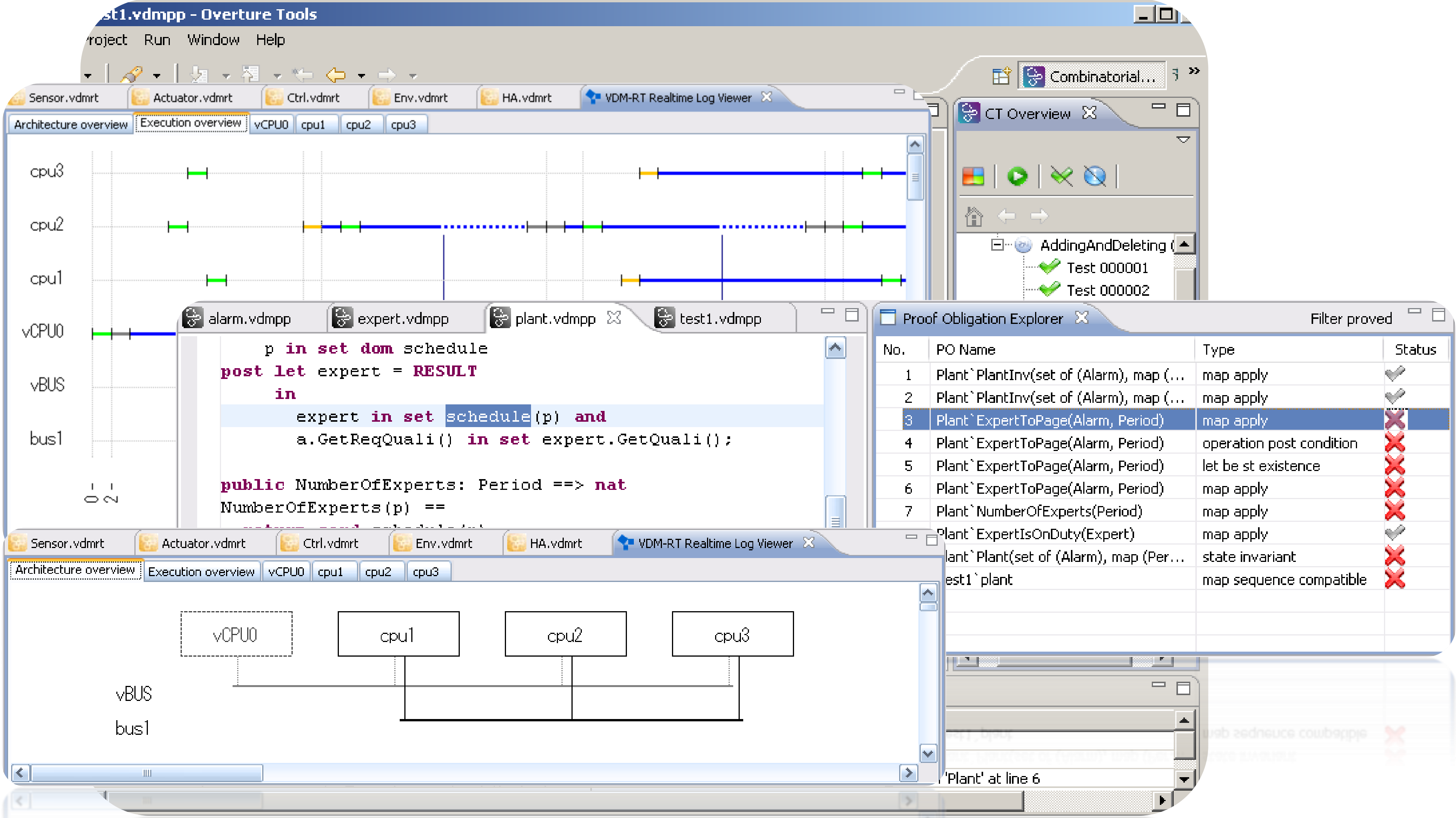
Plant
- alarms
- schedule
- PlantInv(set of Alarm, map Period to set of Expert)
- Period
- ExpertToPage(Alarm, Period)
- NumberOfExperts(Period)
- ExpertIsOnDuty(Expert)
- Plant(set of Alarm, map Period to set of Expert)

Outline

```
=>a
"Alarm{#2, reqQuali:=<Chem>, descr:=\"Tank overflow\"}"
=>expert
"Expert{#4, quali:={<Elec>}}"
=>
```

Interactive console

Other Plug-ins



```
    p in set dom schedule
post let expert = RESULT
     in
       expert in set schedule(p) and
       a.GetReqQuali() in set expert.GetQuali();

public NumberOfExperts: Period ==> nat
NumberOfExperts(p) ==
```

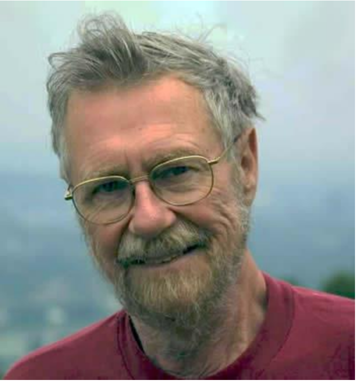
No.	PO Name	Type	Status
1	Plant`PlantInv(set of (Alarm), map (…	map apply	✓
2	Plant`PlantInv(set of (Alarm), map (…	map apply	✓
3	Plant`ExpertToPage(Alarm, Period)	map apply	✗
4	Plant`ExpertToPage(Alarm, Period)	operation post condition	✗
5	Plant`ExpertToPage(Alarm, Period)	let be st existence	✗
6	Plant`ExpertToPage(Alarm, Period)	map apply	✗
7	Plant`NumberOfExperts(Period)	map apply	✗
	Plant`ExpertIsOnDuty(Expert)	map apply	✓
	lant`Plant(set of (Alarm), map (Per…	state invariant	✗
	est1`plant	map sequence compatible	✗

Summary

- What have I presented today?
  - Can mathematics add precision to specifications?
  - What are Formal Methods?
  - Different Formal Methods
  - Where are Formal Methods used?
  - A Guided Tour for VDM-SL modelling
  - Making use of the Overture/VDM tool
- What do you need to do now?
  - Select your VDM case to work with
  - Deliver fully explained and extended specification of the VDM case on Saturday before 16:00 h

Quote of the day

The choice of functional specifications – and of the notations to write them down in – may be far from obvious, but their role is clear: it is to act as a logical 'firewall' between two different concerns. The one is the specification of the engine we would like to have; the other one is the 'correctness problem', ie. the question of how to design an engine meeting the specification. … the two problems are most effectively tackled by … psychology and experimentation for the pleasantness problem and symbol manipulation for the correctness problem.



By Edgar Dijkstra