University Education in Formal Methods — Report on the Minho Experience

FME'97 — Training & Education Workshop
16 September 1997
Graz

J. Nuno Oliveira
University of Minho / INESC
Braga
Portugal

email: jno@di.uminho.pt
http://www.di.uminho.pt/~jno

Abstract

- **Software design** remains among the very few topics which technical university departments often accept to address with little (if any) scientific basis.
- **Teaching** software design with no degree of formalization cannot be accepted but as an insult to current engineering teaching standards.
- For many years the **Formal Methods** (**FM**) community has endeavoured to raise software design from a "handicraft level" to a truly scientific standard.
- This talk reports on the FM teaching experience gathered in the last 12 years at **Minho University** Informatics Department.
- It describes the local evolution of the overall FM teaching strategy and reviews the **main lessons** learnt.

Software Design:

Riddle Guessing or Problem Solving?

Faced with a "folk riddle" such as

> *A football team scores 16 points after losing no match at all and drawing twice as many matches as the ones it won. In which week of the championship are we?*

this "72 year old" uneducated man can but guess an answer by trial-and-error,

won	$drawn$	$week$	$score$	$ok?$
1	2	3	4	no
2	4	6	8	no
3	6	9	12	no
4	8	12	16	$yes!$

while this "12 year old" maths student provides an answer,

$$week \;=\; 12$$

after solving the following mathematical model of the riddle,

$$\left\{\begin{array}{rcl} won + drawn & = & week \\ drawn + 2 \times won & = & 16 \\ drawn & = & 2 \times won \end{array}\right.$$

for unknown $week$.

Will the average "36 year old" software designer address the following "riddle",

> *A bill is a statement of charges for services rendered. A company needs a hierarchical bill manager associating not only charges to service items but also able to unfold such services into sub-bills and so on. Can this hierarchical bill organizer be provided in* ACCESS*?*

by building the following mathematical model of it,

$$\left\{ \begin{array}{rcl} bill & = & service \rightarrow info \\ info & = & charge + subbill \\ subbill & = & bill \\ charge & = & \mathbb{N} \end{array} \right.$$

and solving it for variable $bill$?
Will minimal solution

$$bill \;=\; ((\#bill \times service) \rightarrow charge) \times ((\#bill \times service) \rightarrow \#bill) \times 2^{\#bill}$$

that is, 3 flat tables

$$\begin{array}{|c|c||c|} \#bill & service & charge \\ \hline \cdots & \cdots & \cdots \end{array} + \begin{array}{|c|c||c|} \#bill & service & \#bill \\ \hline \cdots & \cdots & \cdots \end{array} + \begin{array}{|c|} \#bill \\ \hline \cdots \end{array}$$

be reached, subject to a (non-trivial) dataype invariant?

No! the average programmer will probably behave like the 72 year-old person above, in:

- guessing an ‘ad hoc’ data model (*e.g.* ignoring the data type invariant) and
- debugging his/her code in case something goes wrong (*e.g.* the $bill$ browser doesn’t terminate for some of its operations).

Comment

In many teaching contexts software design teaching ignores the **general problem solving** strategy which one is taught at school:

- **understand** your problem
- build a **mathematical** model of it
- **reason** in such a model
- **improve**/upgrade your model, if necessary
- **calculate** a final solution and implement it,

the sophistication of which is only dependent upon the underlying mathematics — from

- simple arithmetics at **primary** school, to
- systems of linear equations, at **secondary** school, then to
- differential/integral equations, in standard calculus, and, in **software calculi**, to
- systems of (recursive) equations on domain spaces.

Can this be accepted in an university setting?

No, by any means!

- Software design remains among the very few topics which technical university departments often accept to address with little (if any) scientific basis.

  $\Rightarrow$ Which civil engineering university department in the world does without (differential/integral) calculus?

- Teaching software design with no degree of formalization cannot be accepted but as an insult to current engineering teaching standards.
- Posterity will regard many of our curricula as typical of some kind of "dark age", or "medieval" stage of software technology.

University Education in Formal Methods

For $\simeq$ 30 years, a world-wide *Formal Methods* (FM) community has endeavoured to raise software design from a "handicraft level" to a truly scientific standard.

FM main aims

- To introduce some notion of "rigour" in software design.
- To resort to mathematics in turning software design into a scientific corpus of knowledge.
- To express and prove properties of software artifacts.
- To calculate software (= **"program from specification"** [Mor90]).
- To automate software calculation.

However…

Quoting [BH95]:

> *The formal methods community is in general very good at undertaking research into the mathematical aspects of formal methods, but not so good at promulgating the use of formal methods in an engineering environment and at an industrial scale.*
>
> *(...) while the number of projects in which formal methods are being employed is growing rapidly, their use is still very much the exception rather than the norm.*
>
> *(...) what makes a formal methods project successful? This is a very subjective question, and to attempt a definitive answer would be ludicrous.*

Yet Another Question:

What makes a **FM education** project successful?

This question is difficult to answer. The author can but report on his own FM teaching experience at Minho University.

FMs at Minho University

The Minho context

- Densely populated region whose industrial context is based on SMEs.
- Great demand for medium-scale information systems.
- Young University ($\geq$ 1973).
- Young Informatics Department ($\geq$ 1977), initially focussed on electronics and control systems with reduced emphasis on software design technology.
- Great freedom to introduce new topics and research areas.

Start off

FMs offered for the 1st time in academic year 1984/85, as a 2-semester optional course intended for 4th year students.

- VDM taught after Jones' famous textbook [Jon80] + [BJ82].
- 6 students (out of $\simeq$ 40) chose the course and got good marks.
- All 6 students adopted VDM in their final (5th) year workstudy in industry.

Evolution (1984/85 $\longrightarrow$ 1994/95)

- The number of students following the FM optional course at 4th year level increased up to $\simeq$ 40%.
- From 1988/89 onwards, FMs became compulsory in the Maths and Computer Science (LMCC) course ($\simeq$ 50 students per year).

Current situation (1996/97)

- Elementary algebraic specification topics offered at 1st year level.
- Process calculi (Henessy-Milner logic, μ-calculus, *etc.*) course offered at 3rd year level.
- FM topics offered in 2 semesters (4th year):
  - Model-oriented formal **specification**
  - **Reification** calculi (including a lab project)

What We Have Learnt (or How We Survived)

As a rule, students

- would find the FM course **interesting**,
- would agree with the **advantages** of formalizing software design

. . . but . . .

- in **"real life"** situations (*e.g.* in the design of a database for a company) they would rather use conventional notations and methods, *e.g.* DFDs (data-flow diagrams), E-R (Entity-Relationship) diagrams, normalization theory, *etc.*

They would ask:

- Do FMs **blend** with conventional methods such as those adopted in other courses?
- Where are the **tools** helping FMs, *e.g.* for RAD (rapid application development) support, invariant preservation proof support, *etc.*?

. . . and further ask:

- what about. . . **user-Interface** design?
- . . . reverse engineering?
- . . . data mining?

FM “tuning” to local demand

We felt that we had to tune our “FM message” to the local demand.
Otherwise, our FM teaching would never leave the school desk. . .

Local demand:

- Information system design
  - RAD tools
    - Interactive applications
    - Visual programming
    - Client-server
  - Database design
    - OO data modelling
    - E-R diagram based design
    - Normalization theory
    - DFD process modelling
  - Services
    - Data mining
    - OLAP
  - Target platforms
    - Oracle
    - SQL
    - Java
    - Windows

From about 5 years ago we felt that many companies were suffering already from the *legacy syndrome*, so we should cater for this area as well.

Teaching strategy ($\geq$ 1987)

On the methodology side:

- Center FM teaching around an abstract theory of **data** generalizing model-oriented specification:

  $\boxed{\Rightarrow \text{ the SETS notation [Oli87]}}$

- Express **data manipulation**, **data transformation**, **data refinement** in terms of SETS (in)equational laws:

  $\boxed{\Rightarrow \text{ the SETS calculus [Oli90, Oli92]}}$

- Blend SETS with **other** calculi:

  $\boxed{\Rightarrow \text{ the REFINEMENT CALCULUS [Mor90]}}$

  $\boxed{\Rightarrow \text{ the FOLD/UNFOLD CALCULUS [Dar82]}}$

- Replace **E-R** diagrams by SETS-expressions [OC93]
- Replace **Normalization**-theory by SETS-calculation [Rod93]

  $\boxed{\Rightarrow \textit{Normalization is Reification!} \text{ [RO97]}}$

- Replace 'ad hoc' relational code design by embedded-SQL code **calculation** from the equations implicit in SETS refinement diagrams (see *e.g.* [Oli97b]).
- Integrate SETS with **other methodologies** taught at other courses (*e.g.* the same project addressed in several courses along the same semester).

About Merging Software Design Calculi:

There is a close link between SETS and the OXFORD REFINEMENT CALCULUS which preserves their relative orthogonality:

- The REFINEMENT CALCULUS is a *weakest pre-condition* [Dij76] (algorithmic) calculus in which *change of representation* is handled by *chosing* coupling invariants.
- SETS' main purpose is that of *calculating* (instead of chosing) such coupling invariants.

Main results:

- SETS has been regularly taught at the University of Minho since 1989.
- The above cited work on relating relational database theory and SETS has been determinant in the way **database design** is taught to Minho's undergraduate students:

Normalized database models are obtained from formal specifications by SETS calculation.

- Final year projects usually involve the full **calculation** of an SQL information server running on top of ORACLE.
- Former students often report on the successful application of the calculus at **industrial** software design level.
- A list of industrial projects which resorted to program calculation in SETS is currently being compiled and will soon become available from the SETS URL:

  `http://www.di.uminho.pt/~jno/html/setshp.html`

- Some former students have set up their own companies in which FMs are (at least partly) used.

Illustration of "the pragmatic spirit"

The following table helps in introducing SETS's **primitive constructs** to "program hackers":

SETS	PASCAL	C	**Description**

$A \times B$	`record`
	`P: A;`
	`S: B`
	`end;`
		`struct {`
		`A first;`
		`B second;`
		`};`
			Records

$A + B$	`record`
	`case`
	`tag: integer`
	`of x =`
	`1: (P:A);`
	`2: (S:B)`
	`end;`
		`struct {`
		`int tag; /* 1,2 */`
		`union {`
		`A ifA;`
		`B ifB;`
		`} data;`
		`};`
			Variant records

B^A	`array[A] of B`	`B ...[A]`	Arrays
$A + 1$	`^A`	`A *...`	Pointers

The Foundations

- Constructive, model-oriented specification based on **set-theory** + a (modest) **category-theoretical** support.
- Data-types (models) regarded as **functors** in the category of countable sets, *e.g.* the *list* functor:

$$
\begin{tikzcd}
A \arrow[r, dotted, no head] \arrow[d, "f"'] & A^\star \arrow[d, "f^\star"'] \\
B \arrow[r, dotted, no head] & B^\star
\end{tikzcd}
$$

- An operator repertoire which is well structured,

  - Operator
    - Set-theoretic — *e.g.* $\mathbf{N} \xrightarrow{\lambda n.n^2} \mathbf{N}$
    - Category-theoretic
      - Implicit — *e.g.* $A \times B \xrightarrow{\pi_2} B$
      - Functorial — *e.g.* $X^\star \xrightarrow{f^\star} X^\star$
      - Transformational — *e.g.* $X^\star \xrightarrow{elems} 2^X$

  and rich in algebraic (categorial) **properties** which are very useful at **calculation** time.

Analogy

The local "FM culture" has followed a path similar to the Irish School of Constructive Mathematics [aA96], for instance:

- their **indexed monoid** construct is a special case of our **functorial** monoid construction [Oli97b];
- we are also working on a *theory of general hierarchical systems* as part of a general model-oriented "sophistication theory" [Oli97a]. For instance, every data-type $\mathcal{F}(A)$ extends hierarchically to the recursive datatype $X \cong \mathcal{F}(A + X)$.

  Examples: Take the finite-mapping datatype $\mathcal{F}(A) \stackrel{\mathrm{def}}{=} B \rightarrow A$, for some B.

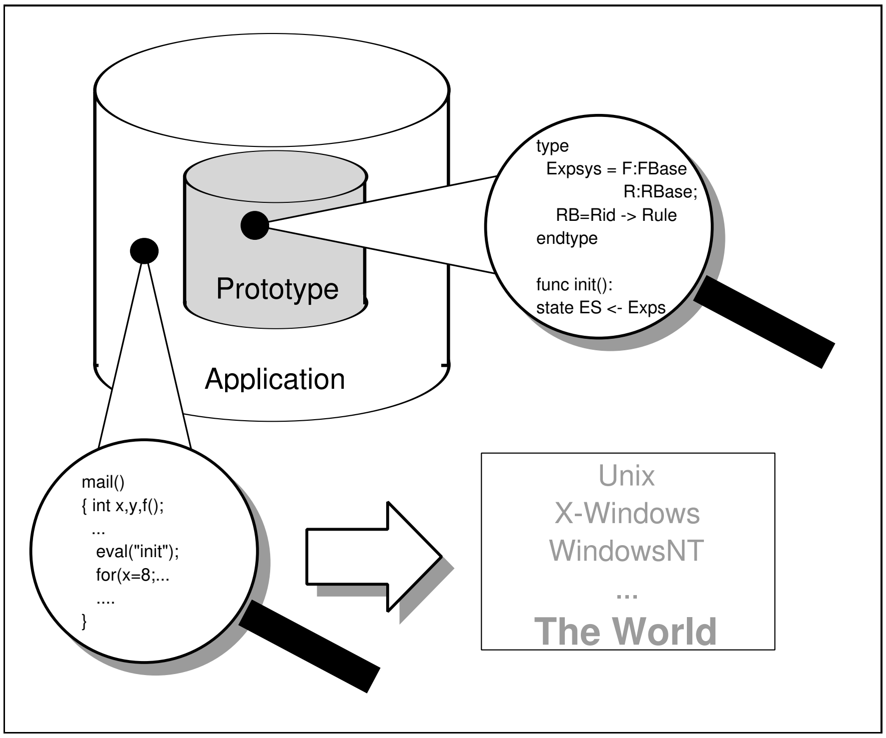
  Then take its hierarchical extension $X \cong B \rightarrow (A + X)$: you have just specified an hierarchical file system, for $A = File$ and $B = FileName$. (Check also the *bill* data-model above.)

  Now take $\mathcal{F}(A) \stackrel{\mathrm{def}}{=} A \rightarrow B$. The same extension will provide you with a specification of the *bill of materials* data-model [OC93].

On the tools side:

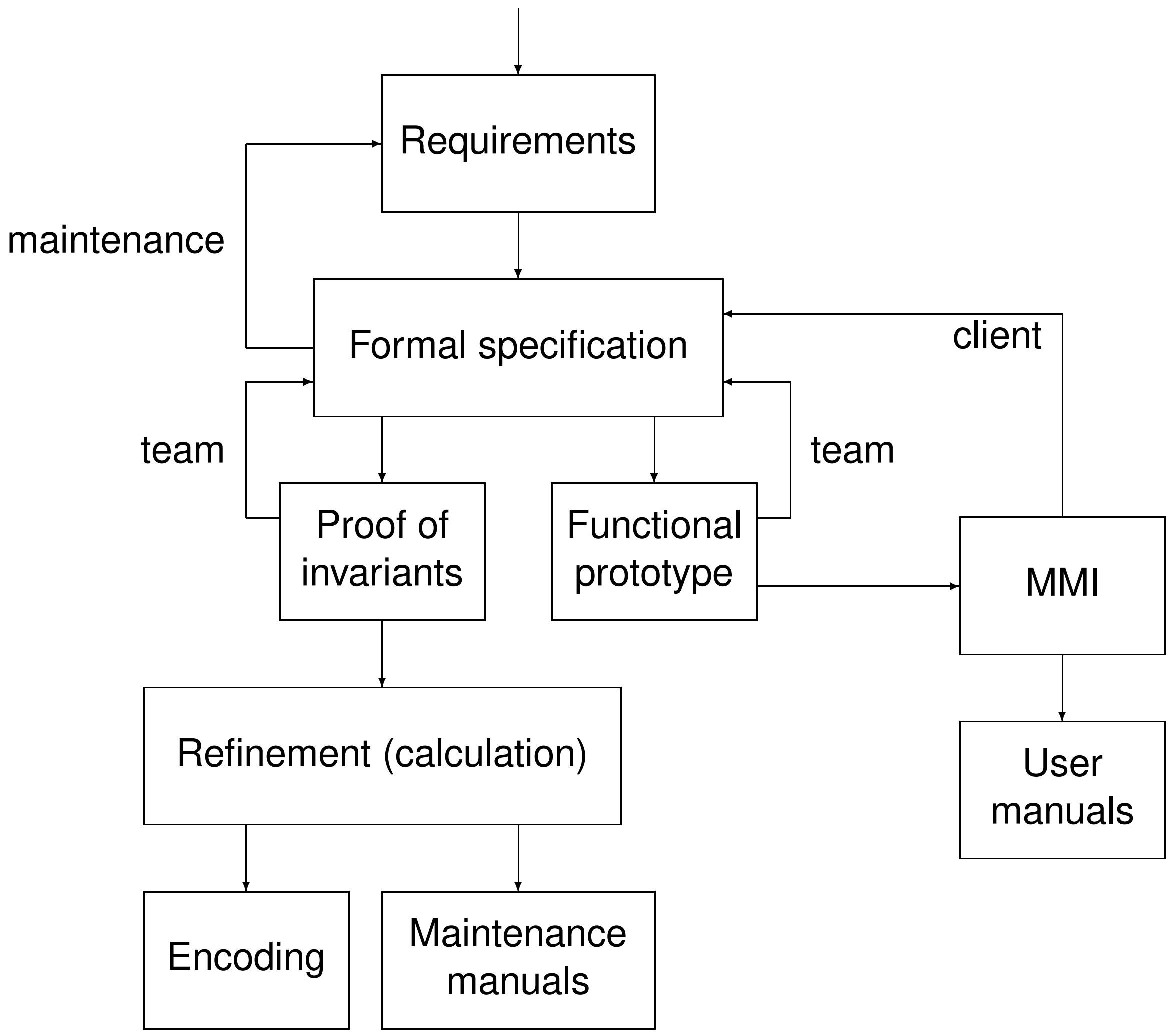
- We have learnt that students adhere to FM animation tools.
- A particular **FM toolkit** has been developed — CAMILA [ABNO97] — intended for large-scale software development using the SETS methodology.

  It currently comprises a rapid-prototyping tool ("animator"), a LaTeX pretty-printer, a LaTeX embedder, a structure-editor etc.
- A SETS **refification calculator** is coming up soon (see the SETS's URL).
- The CAMILA/SETS animator (based on a fairly elaborated X-LISP extension) is available on Unix, MS/DOS and Windows 95 and can be easily interfaced with other tools via C/C++; any CAMILA prototype can be a X-windows server or function like a Windows 95 DLL:



A FM life-cycle sketch

("THOU SHALT TEST, TEST, AND TEST AGAIN" [BH95]):



Current R&D (on the teaching background)

Extending SETS to **process reification** [Bar97].

Final Questions/Comments

Are FMs for software design inherently "difficult"?

- **Software calculi** are semi-polynomial because of the absence of inverses (we have **monoids** instead of **groups**) which rule out cancellation properties and thus the reduction method for solving systems of equations.

$\Rightarrow$ Thus the need for topological (order-theoretic) methods, thus Tarski-Kleene's recursion theory, *etc.*

But this situation is not new in ordinary mathematics, recall *e.g.*

$$x = R \cup R \circ x$$

in a relational setting.

- Need to reason up to some congruence (as a rule, set-theoretic equality is "uncomfortable").
- Is category theory too abstract? Perhaps we still don't know how to **teach** it ...

Are university curricula prepared for FMs?

Perhaps not:

- J. Cuadrado [Cua94] discussed this problem — in the BYTE MAGAZINE (!) — and criticized US curricula for lack of mathematical support for FM-based software design courses.
- Similar defective support can be found elsewhere. In general, ...

... while software design formalization requires disciplines such as *abstract algebra*, *discrete mathematics*, *category theory* and *formal logic*,

... these are rarely found in **traditional** engineering curricula, which are too much concerned with the mathematics of **continuity** to leave room for the mathematics of **discreteness**.

Short term impact of FMs on students

A negative (although unlikely) impact of FM on students is that of **fundamentalism** [BH95]:

"THOU SHALT FORMALIZE BUT NOT OVER-FORMALIZE"
"THOU SHALT NOT BE DOGMATIC"

The expected one is that of a learned but realistic skill [BH95]:

"THOU SHALT NOT ABANDON THY TRADITIONAL DEVELOPMENT METHODS"

In a particular project, there is always some part of it which should be formalized and another one which either cannot be satisfactorily formalized (yet!) or does not deserve any formalization.

Long term impact of FMs on students

Even if

- students end up forgetting about FMs altogether
- they never adopt them **explicitly**

beneficial traces of their rigorous background can normally be found in their professional behaviour, for instance concerning

- terminology (*e.g. invariant*, *pre-condition*, *safety*)
- project organization
- mind-discipline (*e.g.* modularity).

Is there any sociological barrier against FMs?

Perhaps not, but...

- Generalized audio-visual culture shrinks vocabulary and linguistic skills (negative impact on textual analysis)
- Average proficiency in mathematics is levelling down in the whole western world.
- Society wants things "easy and quick" — is the trend for software **certification** and **quality** going to survive?
- Bad examples: pressure from leading companies often results in rushed software with bugs — society seems to acquiesce in such a situation.
- University system is getting too crowded.

References

[aA96] Micheal Mac an Airchinnigh. Some reflections on mathematics education for formal methods. Technical report, University of Dublin, 1996.

[ABNO97] J.J. Almeida, L.S. Barbosa, F.L. Neves, and J.N. Oliveira. Camila: Formal software engineering supported by functional programming. 1997. To be presented at CLAPF-97 — 2nd Latin-American Conference on Functional Programming, Oct. 3–4, La Plata, Argentina.

[Bar97] L. S. Barbosa. *Um Cálculo para Reificação de Processos*. University of Minho, 1997. Ph. D. thesis (in preparation).

[BH95] J. P. Bowen and M. G. Hinchey. Ten commandments of formal methods. *IEEE Computer*, 28(4):56–63, 1995.

[BJ82] D. Bjorner and C. B. Jones. *Formal Specification and Software Development*. Series in Computer Science. Prentice-Hall International, 1982. C. A. R. Hoare.

[Cua94] J. Cuadrado. Teach formal methods. *Byte*, page 292, December 1994.

[Dar82] J. Darlington. Program transformation. In *Funct. Prog. and Its Applications: An Advanced Course*. Cambridge Univ. Press, 1982.

[Dij76] E. W. Dijkstra. *A Discipline of Programming*. Prentice-Hall, 1976.

[Jon80] C. B. Jones. *Software Development — A Rigorous Approach*. Series in Computer Science. Prentice-Hall International, 1980. C. A. R. Hoare.

[Mor90] C. Morgan. *Programming from Specification*. Series in Computer Science. Prentice-Hall International, 1990. C. A. R. Hoare, series editor.

[OC93] J. N. Oliveira and A. M. Cruz. Formal calculi applied to software component knowledge elicitation. Technical Report C19-WP2D, DI/INESC, December 1993. IMI Contract *C.1.9. Sviluppo di Metodologie, Sistemi e Servizi Innovativi in Rete*.

[Oli87] J. N. Oliveira. Refinamento transformacional de especificações (terminais). In *Actas das XII "Jornadas Luso-Espanholas de Matemática"*, volume II, pages 412–417, Braga, Portugal, 4-8 Maio 1987.

[Oli90] J. N. Oliveira. A reification calculus for model-oriented software specification. *Formal Aspects of Computing*, 2(1):1–23, April 1990.

[Oli92] J. N. Oliveira. Software reification using the SETS calculus. In *Proc. of the BCS FACS 5th Refinement Workshop, Theory and Practice of Formal Software Development, London, UK*, pages 140–171. Springer-Verlag, 8–10 January 1992. (Invited paper).

[Oli97a] J. N. Oliveira. Data sophistication is not data refinement. Technical report, DI/INESC, U. Minho, 1997. (In preparation).

[Oli97b] J. N. Oliveira. Sets - a data structuring calculus and its application to program development. Technical report, UNU/IIST, May 1997. Lecture Notes of Course at UNU/IIST, Macau, 5-16 May 1997, 125p.

[RO97] C. J. Rodrigues and J. N. Oliveira. Normalization is data refinement. Technical report, University of Minho, 1997. (in preparation).

[Rod93] C. J. Rodrigues. Sobre o desenvolvimento formal de bases de dados. Master's thesis, University of Minho, 1993. (In Portuguese).