The challenges of software engineering
Logic Programming in the Real World
Some thoughts

Leuven - July 10, 1997

Michel Vanden Bossche-Marquette
Chief Executive

Tel +32 2 759 95 60
Fax +32 2 759 27 60
E-mail mvb@miscrit.be

Background

The changing environment

Society ↔ ICT

✓ Complexity  ✓ Computers
✓ Post-industrial → InfoAge  ✓ Communications
✓ Software

ICT (Information & Communications Technologies)

✓ Computers  technological explosion (mastered)
✓ Communications  technological explosion (mastered)
✓ Software  Instability, confusion, crisis

Computers

Technological explosion (microelectronics)

Asymptotic (= major technological factor)

source: IEEE computer
Communications

- **Digitalisation**
  - Fiber Optic
  - Switching
  - Access
  - SDH (2.4 Gbps), WDM (50+ Gbps)
  - Low cost cell switching fabrics
  - Impact of DSP (xDSL, CATV)

- **Internet**
  - Mass market
  - More PCs than TV sets this year

Software

- **Contradiction**
  - Enormous progress with commodity software (OA, Web, ...)
  - Major problems with software in general
    - Development time and cost are increasing
    - Quality is very poor
    - Maintenance is very expensive (70%)
    - Rigid
    - Additional problems (event-driven, client/server, Web, etc.)

  → **Software Crisis**

  While software is supposed to be malleable, it has become the most rigid part of an information system

Progress "commodity" - Microsoft 97

- Explorer
- Word
- Excel
- PowerPoint
- ActiveX

The Software Crisis

Software easily rates among the most poorly constructed, unreliable and least maintainable technological artifacts ever invented — with the exception, perhaps, of Icarus’ wings.

SOFTWARE: SO BAD, IT CAN ONLY GET BETTER
The Software Crisis

**Large Projects**
- TAURUS: London Stock Exchange: cancelled after 400 M GBP
- Socrates: SNCF, investment >2.000 M FRF, impact on sales
- SABRE 92: Extension Marriott, Hilton, Budget: write-off $160 M
- DENVER: Baggage Handling: late by 18 months, loss $400 M
- France: Projet "Ministère de la Justice": cancelled, 2.000 M FRF
- DB: No payroll during 10 months

**Not forgetting**
- Many smaller projects
- Remanent quality problems (OS/2 Warp, Windows 95, C/S tools, etc.)

At the core of the Information Society

Examples

- 0 gr: Microsoft
- Gram: Dell
- Kilo: General Motors
- Ton: US Steel

Fundamental Problem

Complex systems
- Engineering is mastered
  - conforms to specifications
  - predicted performance
  - within budget
  - very high reliability

Examples
- in line with requirements!
- performance!
- budget!
- reliability!
"Classical" Products

Conceived and built using models based on continuous mathematics
Always a combination of

- Theory
  ✓ models based on physics, ...
  ✓ mathematical expression
  ✓ proof and stability
  ✓ components (theories)

- Practice
  ✓✓✓ applicability of the models
  ✓ tools with required precision
  ✓ expertise (design, tools, ...)
  ✓ components (sub-assemblies)

Precise Corpus of Knowledge

ICT Products

The mathematics on which these products are based is discrete (combinatorial explosion)

- Hardware
  Hardware is based on simple repetitive structures that can be tested exhaustively (but Pentium flaw shows the limit). Mass production is mastered.

- Software
  No repetitive structure and enormous number of states: impossible to do exhaustive testing. Production is much more craft than industry.

Software

Conceived and built using mainly an ad hoc pragmatic approach
The theoretical "corpus" is extremely limited

- Theory
  ✓ variable = position in memory
  ✓ array = contiguous positions
  ✓ proof?

- Practice = craft
  ✓ language, editor, compiler
  ✓ trial and error
  ✓ methodology (?)
  ✓ metric (?)

Examples

- Legal status of software engineering
  ✓ Not one of the 36 engineering professions in USA
  ✓ Tennessee forbids the use of the term "Software Engineering"
  ✓ Essential elements of the engineering profession that are missing:
    - well defined corpus of knowledge
    - formal control of qualification
    - formal process in case of malpractice
    - liability insurance
    - etc.
  ✓ Explicit warranty ("products") ⇔ Explicit no-warranty ("software")
Software Challenges

- **Strategic role**
  - Very low cost of hardware ⇒ Demand increases
  - Supposed to be very flexible ⇒ proliferation of new requirements
  - Software added-value >> Hardware added value

- **Multiple tensions**
  - legacy systems → new platforms
    - + past investments
    - + opportunity
  - new applications
    - - risk, confusion

Hardware - Software

- Client/Server: distributed, relational DB, 4 GLs
- Objects, components, Java
- Web: the "new" Client/Server

Impact

<table>
<thead>
<tr>
<th>Theory</th>
<th>Practice</th>
</tr>
</thead>
<tbody>
<tr>
<td>- no changes</td>
<td></td>
</tr>
<tr>
<td>- RAD (no methodology)</td>
<td></td>
</tr>
<tr>
<td>- Objects (better abstraction)</td>
<td></td>
</tr>
<tr>
<td>- Components (reuse)</td>
<td></td>
</tr>
<tr>
<td>- Turbulence (tools, approaches, ...)</td>
<td></td>
</tr>
</tbody>
</table>

Current Trends

- Hardware - Software
  - Client/Server: distributed, relational DB, 4 GLs
  - Objects, components, Java
  - Web: the "new" Client/Server

Consequences

- **New problems**
  - Much more complex
  - Event-driven systems (instabilities)
  - Strong coupling (inheritance)
  - Impedance mismatch (object - relational)
  - In 5 years: new legacy systems!

- **Packages (SAP, ...)**
  - Solution, but where is the differentiation
  - Software crisis transferred to the vendor
  - New risk: Commercial Of the Self Legacy System

Quality

- **Current approach**
  - ISO 9003: software development, delivery and management
  - Focus is the development process
  - Nothing said about the intrinsic quality of the product

- **Practically**
  - Quality management addresses mainly the practical aspects
  - Approach is "Best Practice"
  - Not bad, but insufficient

- **Example: Windows 95**
  - Beta test with 400,000 users
What is required

- Better balance between practice and theory

Key: respective roles
- Theory: for what, until where?
- Practice: for what, until where?

Not mandatory if problem is "simple"

- 1st industrial revolution
  - 1707: "Root" innovation for the steam engine (Papin)
  - 1712: 1st practical application: pumping (Newcomen)
  - 1765: Basic technology is ready (Watt: condensor, double effect)
  - 1775: 1st mill with adequate precision (Wilkinson)
  - 1854: Pudling iron by steam (steel)

Importance of components and tools

2d industrial revolution
- 1831: Electro-magnetic induction (Faraday)
- 1864: Electro-magnetic field theory (Maxwell)
- 1869: Dynamo (Gramme)
- 1876: Telephone (Bell)
- 1886: Electro-magnetic waves (Hertz)
- 1896: Radio (Marconi)

Importance of components, tools and theory
Theory - Role #2

- Experts
- Users
- "Natural" Knowledge
  - global
  - unformal
  - default knowledge

- Implementors
- "Artificial" Knowledge
  - detailed
  - low level
  - coded
  - complete

Change

"universe"

Too slow (unadapted time constant)

"Pure" Knowledge (declarative)

Transformation

Operational System
  (Many low-level details)

Research

- Computer Science

  - Program = theory in an adequate logic
  - Computing = deduction from that theory

  - Definition of the problem
  - Interpretation of the "intentions" of the programmer
  - Logical part (a restricted theory)
    - Logic programming (theory, axioms, 1st order logic)
    - Functional Programming (equations, theory of types)
  - Control part (efficiency of the implementation)

- Real-life

  - Nearly never (Z, B excepted for safety critical systems)

Software Industry

- In general

  - Yawning chasm between Practice and Computer Science
  - Even for advanced projects
  - Percolation very slow

- Hardware

  - Fundamentally different
  - Very fast transfer from the lab to the industry

- Why?

  - Culture and Education (in general, but some exceptions)
  - Added complexity from legacy
  - Intrinsic complexity
  - Not a closed world: interaction with humans
### Additional Problem

- Instability and confusion

```
exponential complexity
```

### Information & Communication Society

- The speed of interaction in the society is continuously increasing.
- The industries of the first 3 "ages" are stabilized (oligopolies).
- The speed of interaction in the information society is "terminal".
- The information society is entering into turbulence and starts oscillating.
- Turbulence results from the continuous creation of immaterial artifacts.
- Innovation (mainly "immaterial") = major opportunity-risk.

### Society Duration, Interaction, Power, Technology

<table>
<thead>
<tr>
<th>Society</th>
<th>Duration (years)</th>
<th>Interaction (km/h)</th>
<th>Power</th>
<th>Technology</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agrarian</td>
<td>3000-5000</td>
<td>5-15</td>
<td>Land</td>
<td>Tradition</td>
</tr>
<tr>
<td>Industrial</td>
<td>300-500</td>
<td>50-150</td>
<td>Capital</td>
<td>Science</td>
</tr>
<tr>
<td>Post-Industrial</td>
<td>30-50</td>
<td>300 - 900</td>
<td>Structure, coverage</td>
<td>Inverse economics</td>
</tr>
<tr>
<td>Information</td>
<td>3-5</td>
<td>3,000,000+</td>
<td>Communication</td>
<td>Inference, creativity</td>
</tr>
</tbody>
</table>

### The good news

- Exponential Power at declining costs

```
<table>
<thead>
<tr>
<th>Mips</th>
<th>#users</th>
<th>Mips/user</th>
<th>Cost</th>
<th>Cost/user</th>
<th>Cost/Mips</th>
</tr>
</thead>
<tbody>
<tr>
<td>70s - 370</td>
<td>2</td>
<td>100</td>
<td>0.02</td>
<td>3,000</td>
<td>30</td>
</tr>
<tr>
<td>80s - VAX</td>
<td>1</td>
<td>50</td>
<td>0.02</td>
<td>750</td>
<td>15</td>
</tr>
<tr>
<td>1991</td>
<td>30</td>
<td>1</td>
<td>30</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>1994</td>
<td>150</td>
<td>1</td>
<td>150</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>1997 mono</td>
<td>500</td>
<td>1</td>
<td>500</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>2000 clusters</td>
<td>20,000</td>
<td>250</td>
<td>2500</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>
```
The Possibility to leverage that Power

1. Problem: COBOL, FORTRAN, C, ...
2. Next Generation Software
3. Semantic Gap
4. Power

Low-level efficiency

Better abstraction

Pro- or abstraction

1 Mips

500 - 20,000 Mips

Emergence of some stability

- Past
  - Mainframe: Stability, but monopoly prices (IBM, etc.)
- Unix
  - Openess: Lower costs, but...
  - Balkanisation: Sun/Solaris, HP/UX, SCO, etc
- Microsoft - Windows NT
  - Proprietary: But commodity pricing (volume)
  - Stability: Competition but on standardized equipment
- Network
  - TCP/IP: The winner: global connectivity and services
  - Transport: Various (Gigabit Ethernet, ATM, WDM, ...)

Action

- Power
- Stability
- NT - TCP/IP

Better Balance

- Theory
  - Declarative (Logic) Programming
  - Discrete Mathematics
  - Formalization of complex and evolving knowledge
  - Theory to deduce and prove
- Pragmatism
  - Object Oriented Programming
  - Better abstraction
  - Adequate for GUI, Network, Security, etc.
  - Possibility to leverage components (ActiveX, JavaBeans)
**Respective Roles**

- Logic Programming
- Specification Transformation

- 80% "Simple" Part OOP
- 20% Difficult

**How to Implement**

- "0 defect" kernel
- GUI
- ergonomics
- commodity software
- Web
- COM, Java
- C++, Java, ActiveX, ...

**Pedagogical Example**

- Surface
  - Microsoft plumbing
  - VC++, MFC, ActiveX

- Fundamental aspects
  - Accounting Ontology
  - Logic (Prolog)

- Middleware: mapping logic-C++
Example - LogIHCS

- Tarification and Billing (Hospitals)
  - Complex regulation (social security)
  - 4000+ “codes” and 1000+ “rules”
  - Very complex inter-relations
  - Continuously changing
  - Very difficult to implement and maintain in COBOL, C, 4GL, etc.

- Mission Critical's approach
  - Codes and rules in a dedicated formal logic language
  - OOP where adequate
  - Billing = inference engine
  - Tools: browsers, local heuristics, explanation, etc.
  - Scientific partnership (B. Le Charlier, Y. Deville)

Electronic Documentation ...

Win32 Client

Start your business rules engine
Patricia B. Seybold

Are you still entangling the rules of your business in application logic or database triggers? You're in big trouble. There's no way you can change those systems into flexible applications that meet the rapidly changing needs of your business.

Instead, you should engage the logic - have it handled by your applications code. All business rules should be implemented as code. However, to do so requires a different kind of code base. A business rule engine is encapsulated in a database or application server that can be used to capture, store, and manage business rules. If you're not using such an engine, your company is losing money. Here's why:

1. Rule engines make business managers, not IT staff, own business rules and change them. Without an engine, you don't have a central place to store business rules. Without an engine, you can't separate the business logic from the application logic. What's worse, you can't change business rules without a business expert. With an engine, business rules can be separated from the rest of the application logic, making it easier for business managers and developers to manage changes. And since the business rules are stored in a database, they can be changed without affecting the application code.

2. The right business rule engine lets you capture, store, and manage business rules. A business rule engine should provide a way to capture business rules, store them in a database, and manage them. The engine should allow you to test business rules and ensure they work as intended. It should also provide a way to create rule sets and manage rule versions. In addition, the engine should offer a way to track rule changes and provide documentation for any rule versions.

3. The right engine lets you create and manage rule sets. A business rule engine should provide a way to create rule sets and manage them. The engine should allow you to create new rule sets, manage existing ones, and delete them when they're no longer needed. The engine should also provide a way to import and export rule sets, allowing you to share them with other applications or organizations.

4. The right engine lets you manage rule changes and provide documentation for any rule versions. A business rule engine should provide a way to manage rule changes and provide documentation for any rule versions. The engine should allow you to track rule changes and provide documentation for any rule versions. The engine should also provide a way to manage rule versions, allowing you to create new versions and manage existing ones.

5. The right engine lets you create and manage business rules. A business rule engine should provide a way to create and manage business rules. The engine should allow you to create new rules, manage existing ones, and delete them when they're no longer needed. The engine should also provide a way to import and export rules, allowing you to share them with other applications or organizations.

6. The right engine lets you test business rules and ensure they work as intended. A business rule engine should provide a way to test business rules and ensure they work as intended. The engine should allow you to test rules individually or as part of a rule set. The engine should also provide a way to test rule versions, allowing you to test new versions before they're deployed.

7. The right engine lets you create and manage rule sets. A business rule engine should provide a way to create and manage rule sets. The engine should allow you to create new rule sets, manage existing ones, and delete them when they're no longer needed. The engine should also provide a way to import and export rule sets, allowing you to share them with other applications or organizations.

8. The right engine lets you manage rule changes and provide documentation for any rule versions. A business rule engine should provide a way to manage rule changes and provide documentation for any rule versions. The engine should allow you to track rule changes and provide documentation for any rule versions. The engine should also provide a way to manage rule versions, allowing you to create new versions and manage existing ones.

9. The right engine lets you create and manage business rules. A business rule engine should provide a way to create and manage business rules. The engine should allow you to create new rules, manage existing ones, and delete them when they're no longer needed. The engine should also provide a way to import and export rules, allowing you to share them with other applications or organizations.

10. The right engine lets you test business rules and ensure they work as intended. A business rule engine should provide a way to test business rules and ensure they work as intended. The engine should allow you to test rules individually or as part of a rule set. The engine should also provide a way to test rule versions, allowing you to test new versions before they're deployed.

The right business rule engine is a critical component of any business system. It allows you to capture, store, and manage business rules in a database or application server, making it easier for business managers and developers to manage changes. The engine should provide a way to test business rules and ensure they work as intended, allowing you to track rule changes and provide documentation for any rule versions. In addition, the engine should provide a way to create and manage rule sets, allowing you to create new rule sets, manage existing ones, and delete them when they're no longer needed. The engine should also provide a way to import and export rule sets, allowing you to share them with other applications or organizations.
Theory - Opportunities

- The market begins to understand the software challenge
- Business Engineering requires new Information Systems
- Quality concerns (TQM, ...)
- New platforms (Windows NT, Web, Java, ...)
- Exponential power (PentiumPro at commodity prices)
- Many mission-critical and business-critical problems
- Global Information Society: mainly a software problem
- R&D results
- Long-term, as opposed to the short-term software fashion

Theory - Threats

- Legacy systems
- Conservatism
- Intellectually challenging, in opposition to mass market
- Complex problems ⇒ difficult
- Necessity to understand the application domain ⇒ difficult
- Very short life cycle of IT products
- Upstream dominance of the USA
- Logic = "academic" image
- Lack of clear demonstration of the relevance

Approach Mission Critical

- Clear demonstration of the relevance
- Mercury (Melbourne)
  - pure
  - programming in the large
  - declarations: polymorphic types, modes, determinism
  - high performance
- Industrialisation of Mercury (ARGo, within ESPRIT)
  - abstract interpretation (destructive updates)
  - environment
  - methodology
  - demonstration of the relevance

Engineers of the "0 gram" industry

Users

- Knowledge
  - non-formal
  - global
  - natural

"Formalises"

- Knowledge
  - formal
  - global
  - high level

Implementors

- Knowledge
  - formal
  - detailed
  - low level

"production" continuum
reliability, performance, adaptability

"knowledge" continuum
incremental use of knowledge
"ontologies" and "learning organisation"