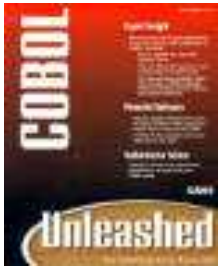
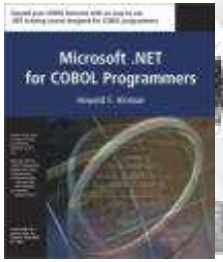
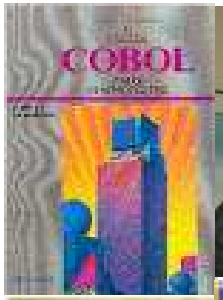
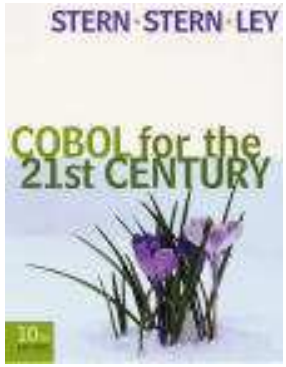
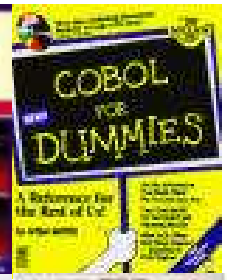
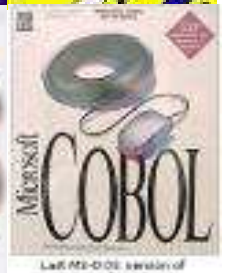












IDENTIFICATION DIVISION.

PROGRAM-ID.

Cobol as a research theme.

AUTHOR.

Ralf Lämmel

on behalf of the IMSE dept.

at the Free University Amsterdam.

vrije Universiteit amsterdam

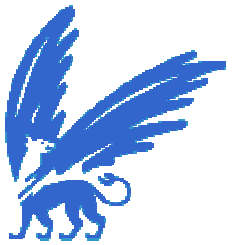
My friend, the smart Java aficionado mumbles ...

Give me one good reason to pursue research on the Cobol language and its applications!

Worse than that ...

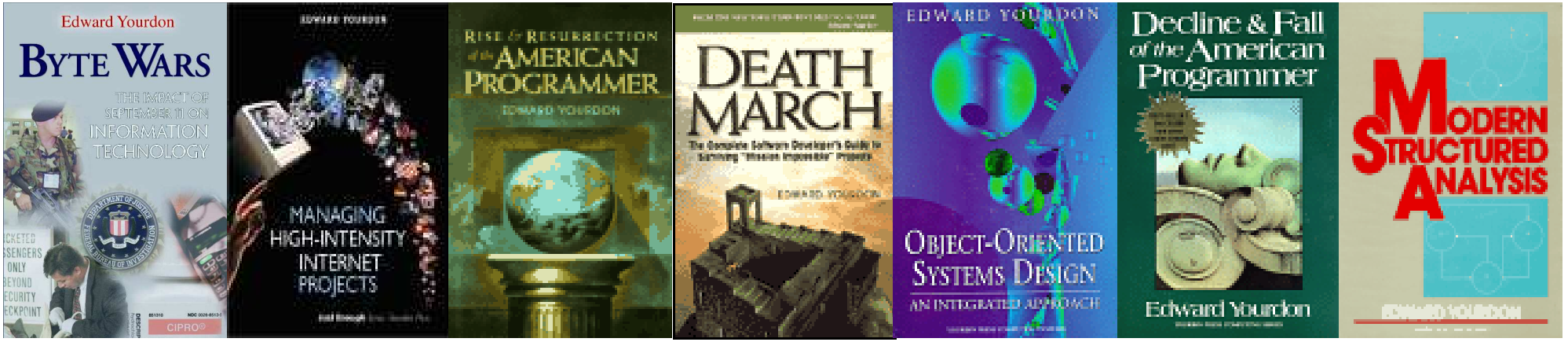
Don't you know about the opinions of the leaders in your area? Such as:

- **Ed Yourdon**
- **Edsger Dijkstra**



Ed Yourdon in Rise and Resurrection of the American Programmer "*... then it's only a matter of time before all the existing COBOL programmers die of old age. Hopefully, the legacy COBOL code which has been estimated at 18-200 billion lines of code will die with them. Or, if it must be kept alive, maybe it will all be outsourced to some part of the world where COBOL maintenance programming is considered a pleasant alternative to growing rice or raising pigs.*"

Don't miss Edmund C. Arranga: *An Open Letter to Ed Yourdon*
http://objectz.com/cobolreport/archives/TCR_ayourdon.htm



Alan M. Turing
The First Society in Computing

Edsger W. Dijkstra, 18 June 1975
How do we tell truths that might hurt?

“ ... The use of COBOL cripples the mind; its teaching should, therefore, be regarded as a criminal offence. ...”

Dijkstra was the 1972 recipient of the Association for Computing Machinery's Turing Award.

Quotes – some of them being more adequate
http://www.sysprog.net/quotcob.html

Dijkstra said:
"*... its [Cobol's] teaching should [...] be regarded as a criminal offence.*"

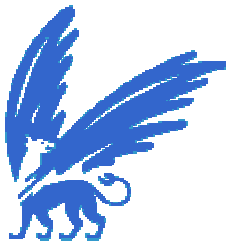
Dijkstra ***did not*** say:
"*... **research** on Cobol should be regarded as a criminal offence.*"

(I manually checked all the 66 hits*!*)

So Cobol research is not a banned subject, not even in the view of free speech.

My friend agrees, but (s)he emphasises ...

Give me *one good reason* to pursue research on the Cobol language and its applications*!*

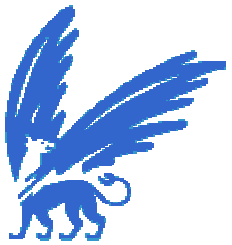


***So here is one good reason:* suppose business computing deserves attention from academia, then Cobol-related research is a good idea.**

Gartner Group

- 75% of all business data is processed in COBOL.
- There are between 180 billion and 200 billion lines of COBOL code in use worldwide.
- 15% of all new applications (5 billion lines) through 2005 will be in COBOL.

So the aforementioned precondition holds.
Unless Cobol and Cobol assets wouldn't need help?!



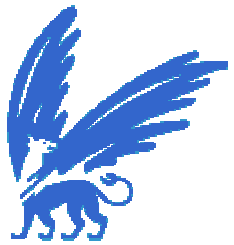
In mathematical terms: **The amount of research dedicated to a programming language should be proportional to (should correlate with) the usage of the language in reality.**

Again, Google science as an oracle:

Search "International Conference on Software Maintenance" and ...

- ... C++ returns 2,150 hits.
- ... Cobol returns 681 hits.
- ... C# returns 64 hits.
- ... Haskell returns 135 hits.

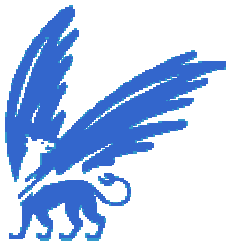
So Cobol deserves more attention.



Here is another good reason to pursue Cobol research: Been there. done that.

Dynamic loading	~ Cobol's CALL?
Meta data	~ ID and ENV DIVISION?
Web services	~ CICS transactions?
Aspects	> Cobol's USE?
ADO.NET	< Embedded SQL Preprocessing?
...	

Languages should help each other.
This makes them all better.



The future of data access?

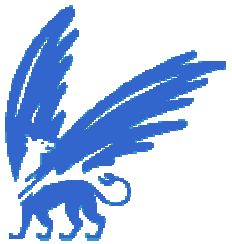
String-based data access with ADO.NET

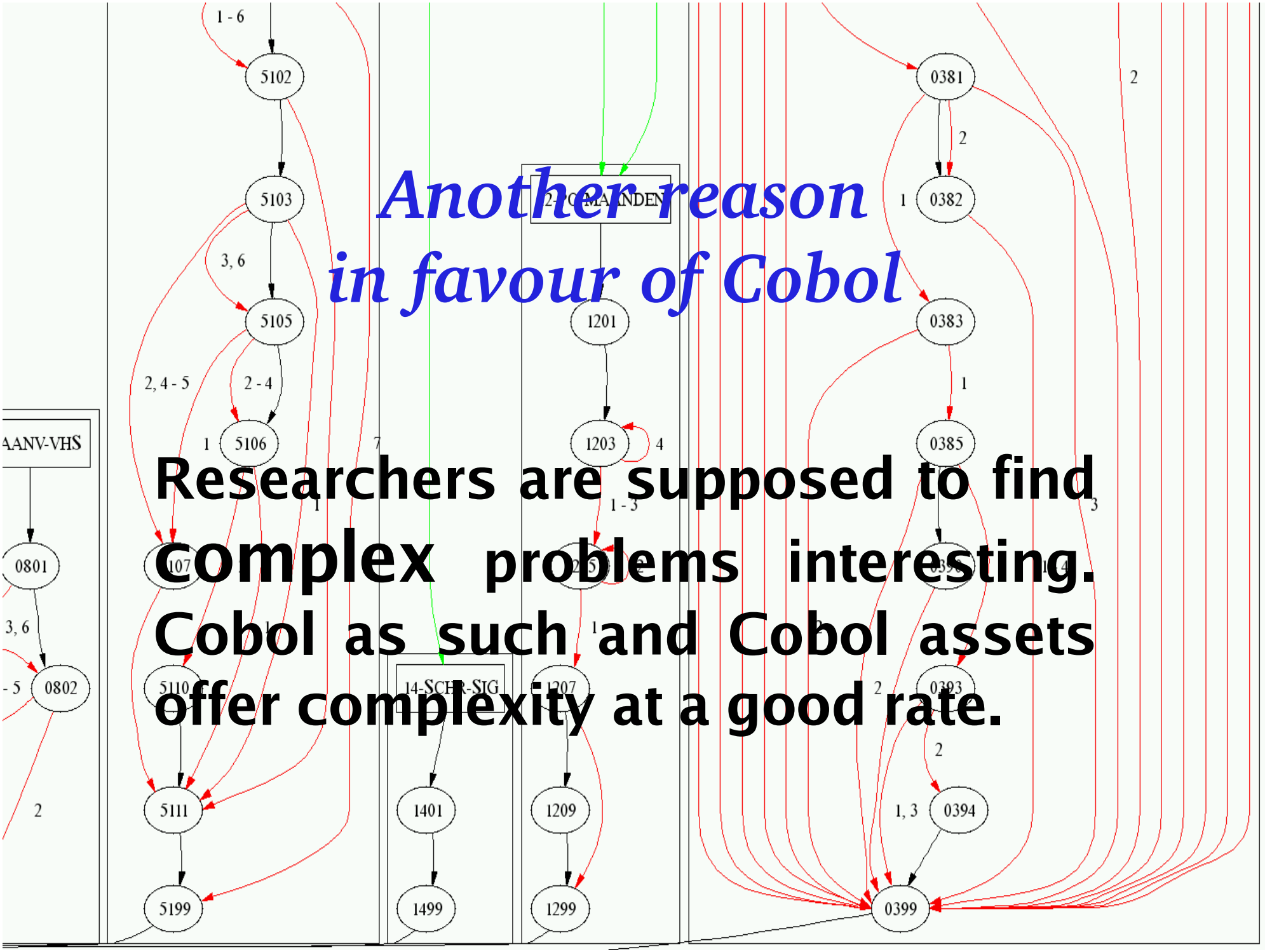
```
MOVE "SELECT FirstName, LastName, HomePhone FROM Employees" TO MY-STRING.
INVOKE CLASS-SQLCOMMAND "NEW" USING MY-STRING CONNECTIONOBJ RETURNING SQLCOMMANDOBJ.
INVOKE SQLCOMMANDOBJ "ExecuteReader" RETURNING DATAREADEROBJ.

MOVE B"1" TO MY-BOOLEAN. *>Set to True to begin with
MOVE ZEROS TO MY-COUNT.
*> Primer read to see if we have data to read back
INVOKE DATAREADEROBJ "Read" RETURNING MY-BOOLEAN. *> True if data, False if none
PERFORM WITH TEST BEFORE UNTIL MY-BOOLEAN NOT EQUAL B"1"
  INVOKE DATAREADEROBJ "GetString" USING 0 RETURNING FIRSTNAME
  INVOKE DATAREADEROBJ "GetString" USING 1 RETURNING LASTNAME
  INVOKE DATAREADEROBJ "GetString" USING 2 RETURNING HOMEPHONE
  ADD 1 TO MY-COUNT
  DISPLAY "Record: ", MY-COUNT, " NAME: ", FIRSTNAME, " ", LASTNAME, " ", HOMEPHONE
  INVOKE DATAREADEROBJ "Read" RETURNING MY-BOOLEAN *> True if data, False if none
END-PERFORM.
```

Source: Data Access with ADO.NET (by Rick Malek)

http://www.csharphelp.com/archives2/archive404.html





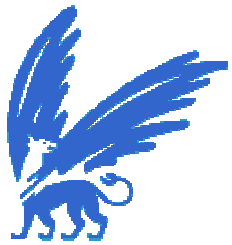
Another reason in favour of Cobol

Researchers are supposed to find complex problems interesting. Cobol as such and Cobol assets offer complexity at a good rate.

What's complex about Cobol?

- Cobol's grammar is the biggest.
- Cobol's constructs are overloaded.
- Cobol assets are ***very*** heterogeneous.
- Cobol assets are not amenable to 100% formal reasoning.
- Data and dialogue models in information systems are huge.

This is not meant to complain.
This complexity is needed or at least justifiable.
Some sort of complexity is the scientist's food anyhow.



Yet another reason in favour of Cobol

– Cobol makes happy –

Who is who at the Free University Amsterdam
(not everyone below is working full-time on Cobol matters)

Steven Klusener | Jan Kort (UvA) | Ralf Lämmel | Niels Veerman | Ernst Verhoeven | Chris Verhoef | Vadim Zaytsev

We gratefully acknowledge collaboration with other parties in Amsterdam (and elsewhere).

software improvement group

CWI

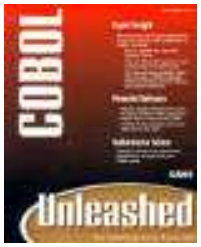
PinkRoccade

vrije Universiteit amsterdam

Typical articles in our research group

- Automatiseren onderhoud complex maar lucratief (beware: Dutch!)
- Architectural Modifications to Deployed Software
- Cracking the 500-language problem
- Generation of Components for Software Renovation Factories
- Object-Oriented COBOL: Concepts & Implementation
- Revitalizing modifiability of legacy assets
- Scaffolding for Software Renovation
- Semi-automatic grammar recovery
- The Amsterdam toolkit for language archaeology
- What does aspect-oriented programming mean to Cobol?
- **...**

Please find some copies at the PR desk!



vrije Universiteit amsterdam

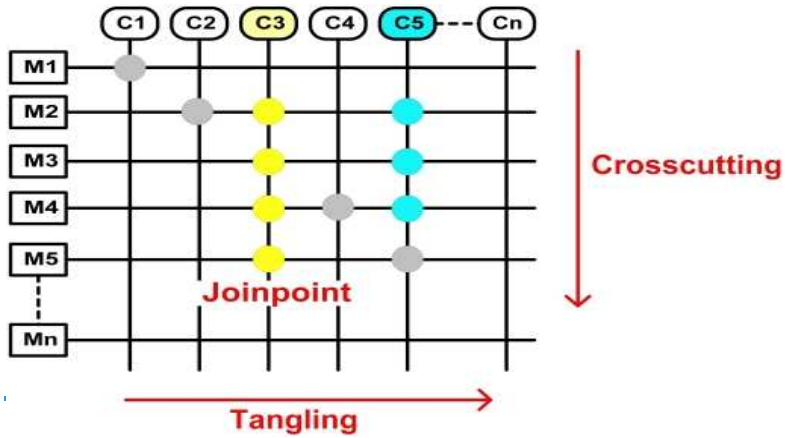
Activities selected for this symposium

- Program restructuring, e.g., GO TOs elimination for Cobol
- Grammar recovery and language reference reverse engineering
- Aspect-Oriented Cobol for OO-ish and non-OO-ish Cobol

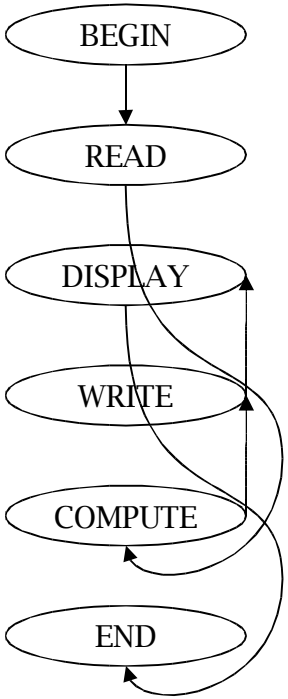
Grammar nerds at VU ...

GO TOs ...

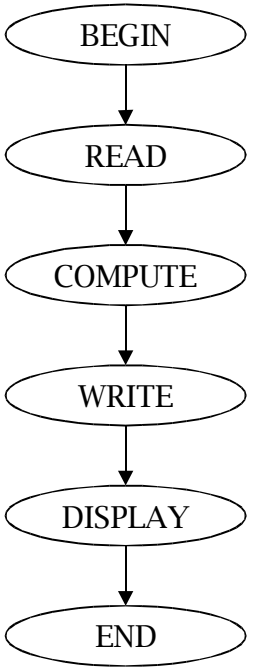
AOP for business programmers ...



Research activity – Cobol restructuring



Automated Spaghetti Elimination



GO TOs ...

The VUA theme on Cobol restructuring

- Improve structure, e.g., eliminate GOs
- Rewriting approach to transformation
- Basic restructuring rules + strategy
- Case studies in the millions LOC range
- 60-80% goto-free code

The ASF+SDF Meta-Environment is used for the implementation of analyses and transformations.

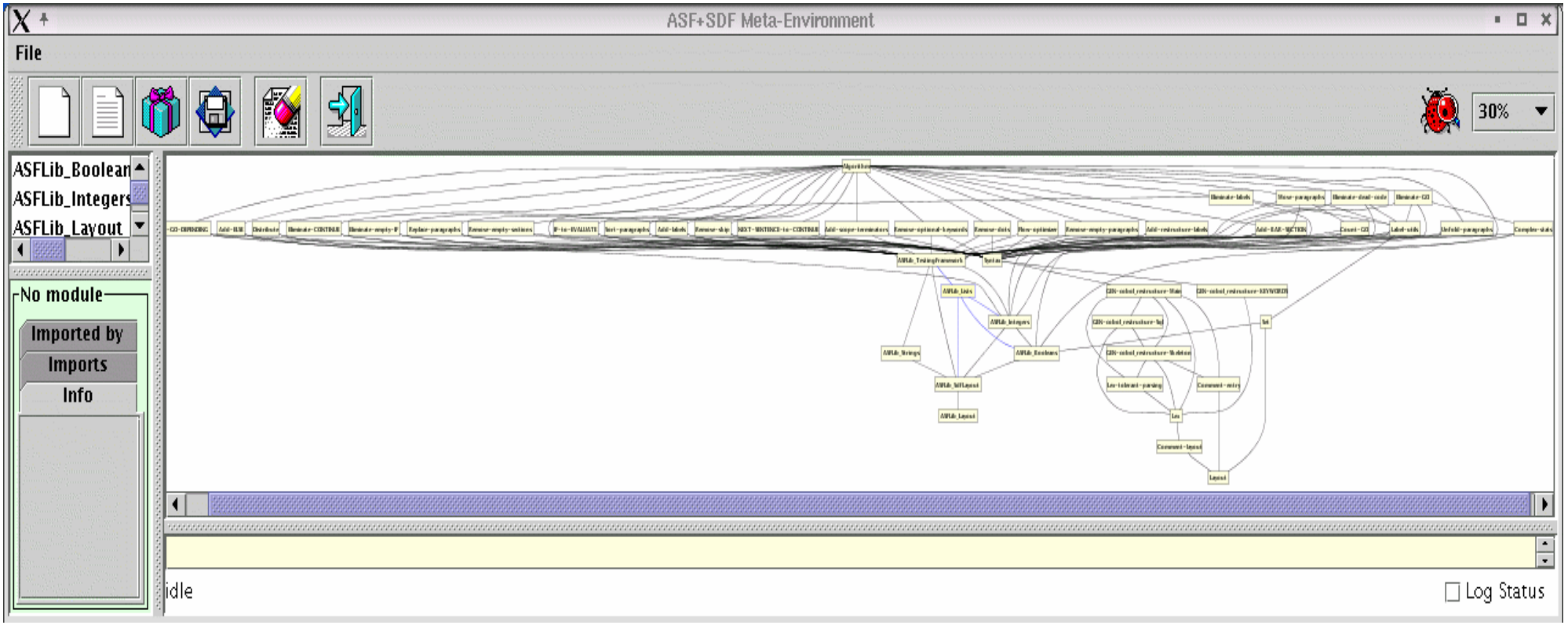
GO TOs ...

Thanks to our CWI colleagues!

CWI

The ASF+SDF Meta-Environment

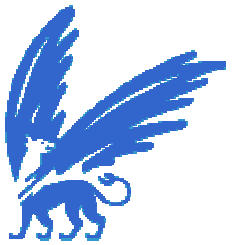
- ASF – Algebraic Specification Formalism (for rewriting)
- SDF – Syntax Definition Formalism (for concrete syntax)
- Parser technology (GLR), Pretty printing technology
- Coordination architecture (toolbus)
- ...



Research issues in automated transformation

- Overall issues
  - Productivity
  - Reuse
  - Robustness
  - Scalability
  - Customisation

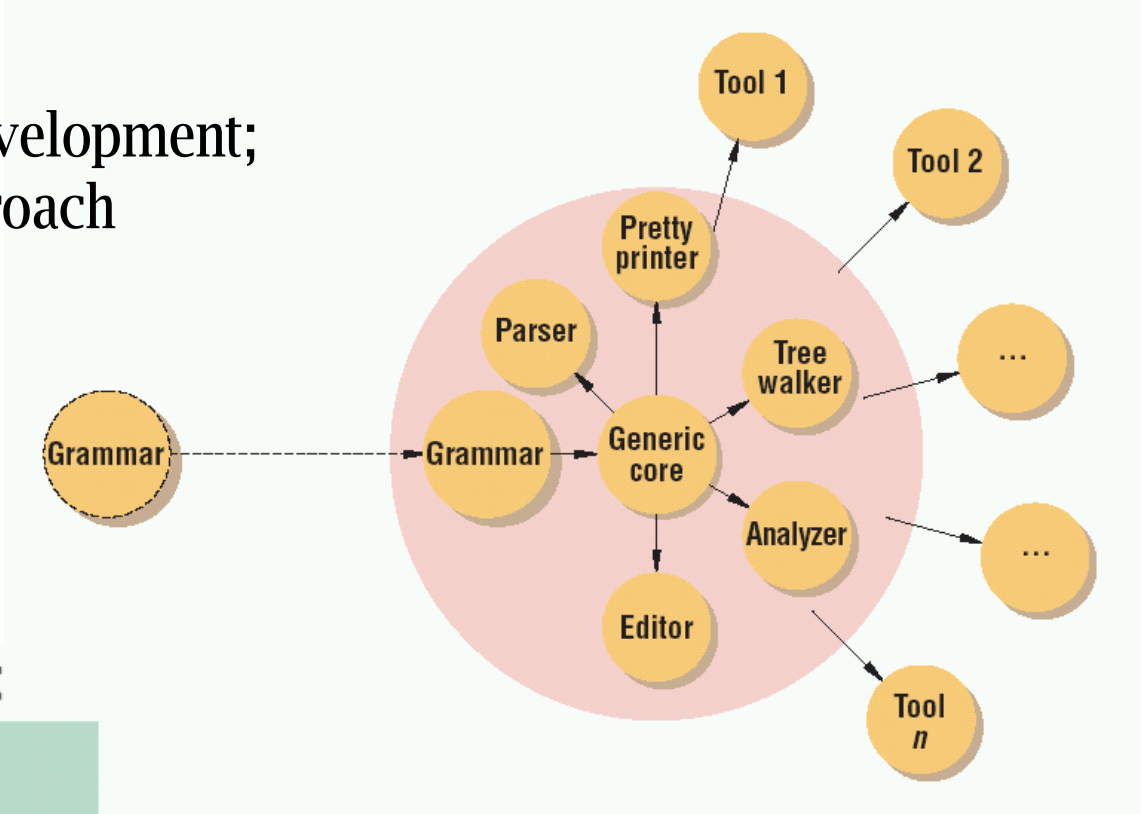
- Technical issues
  - Parsing everything
  - Traceability through phases
  - Pretty printing nicely
  - Generic traversal
  - Pre- and postconditions
  - Language independence



Next research activity

– Grammar recovery and language reference reverse engineering –

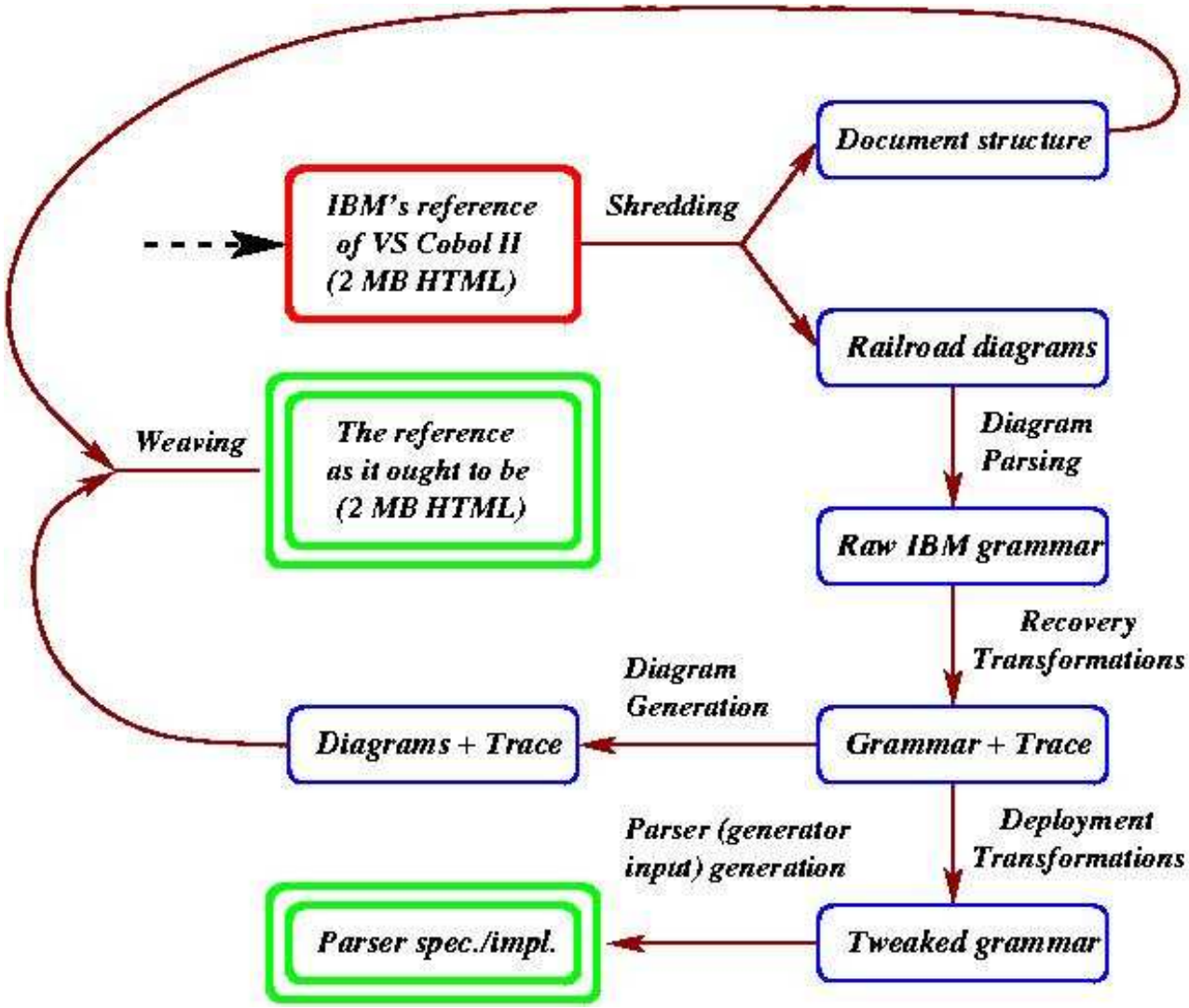
Motivation:
effort shift for renovation tool development;
traditional versus innovative approach



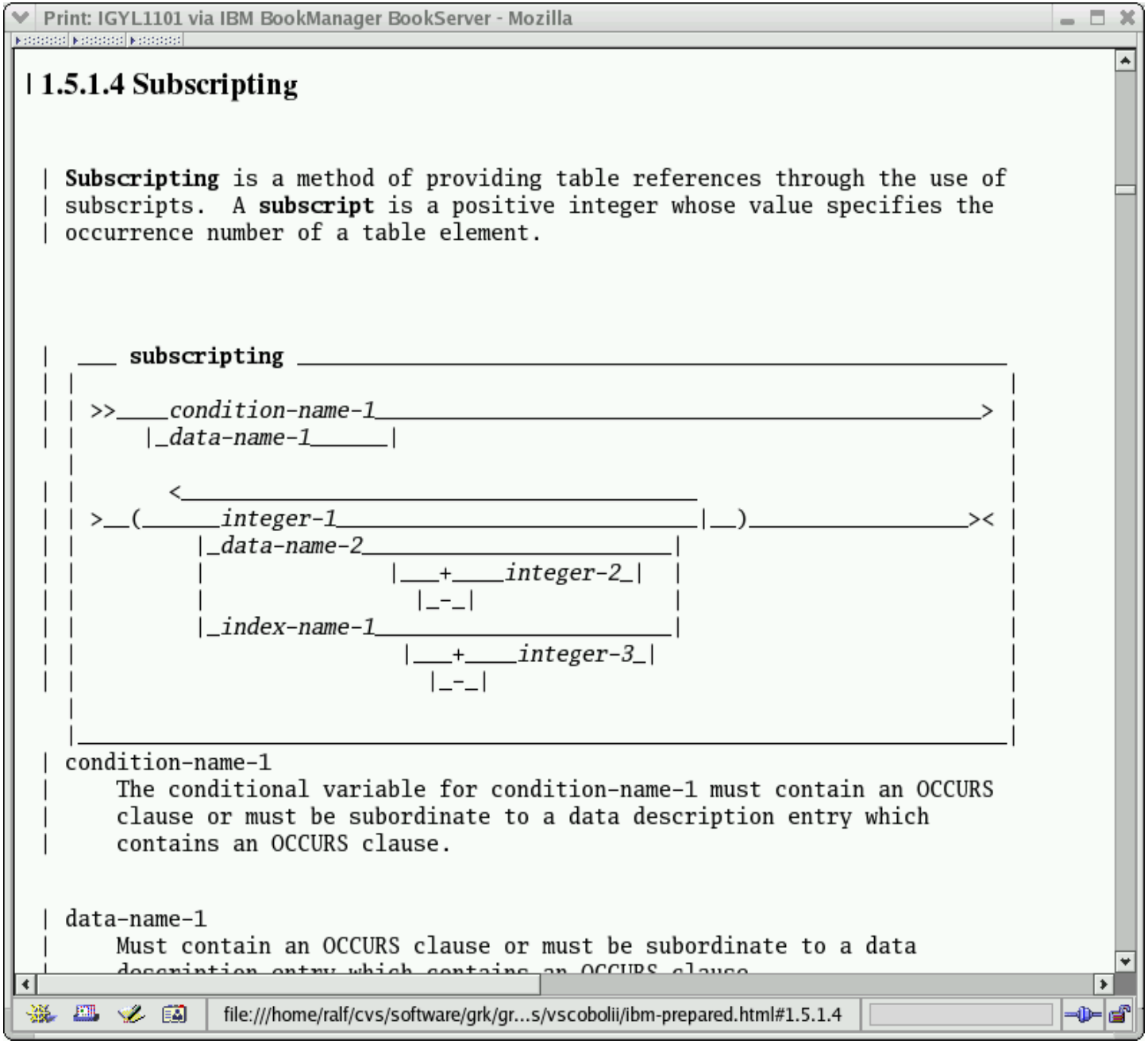
Effort for the 8574 Project

Phase	Effort
Extract the grammar	Two weeks
Generate the parser	One day
Build six tools	Five days
Assemble all the components	One hour
Total	Three weeks

The IBM case study in grammar stealing



The original IBM reference



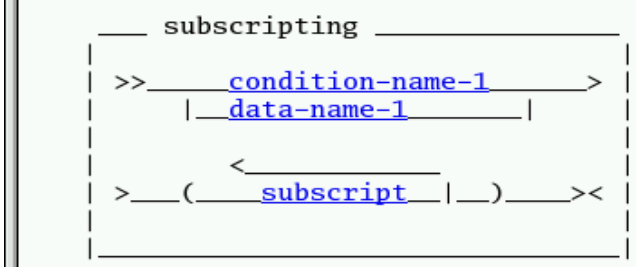
The tweaked IBM reference

Print: IGYL1101 via IBM BookManager BookServer - Mozilla

Disclaimer: *This is not an official IBM document. Make sure that you understand the full disclaimer text as a prerequisite to using the present document.*

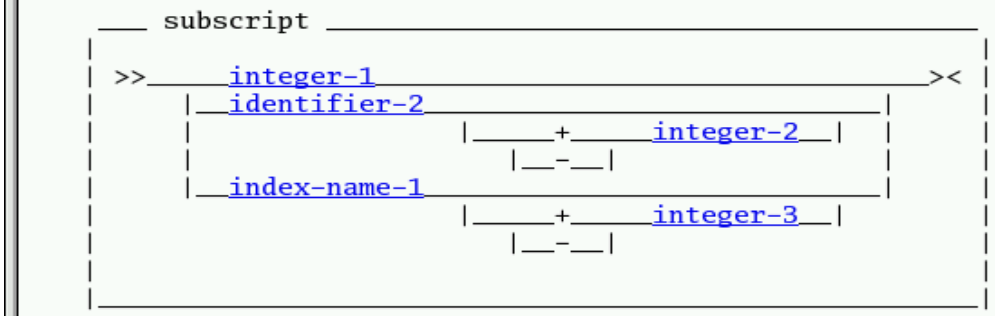
1.5.1.4 Subscripting

Subscripting is a method of providing table references through the use of subscripts. A **subscript** is a positive integer whose value specifies the occurrence number of a table element.



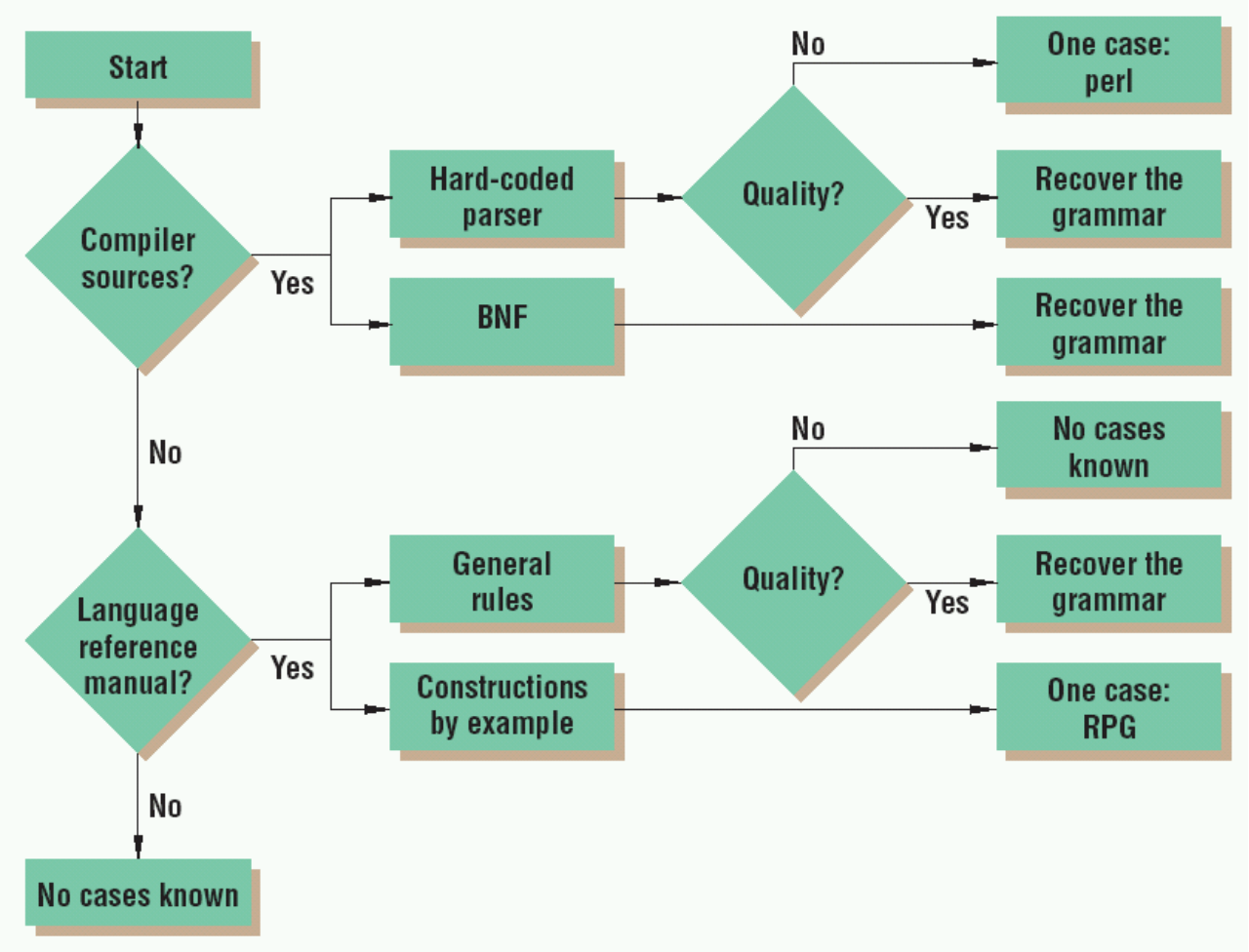
@@ Diagram subscript EXTRACTED from diagram subscripting.
@@ Identified phrase as referred to elsewhere.

@@ Diagram subscript ADAPTED.
@@ Enabled identifiers instead of plain data names.

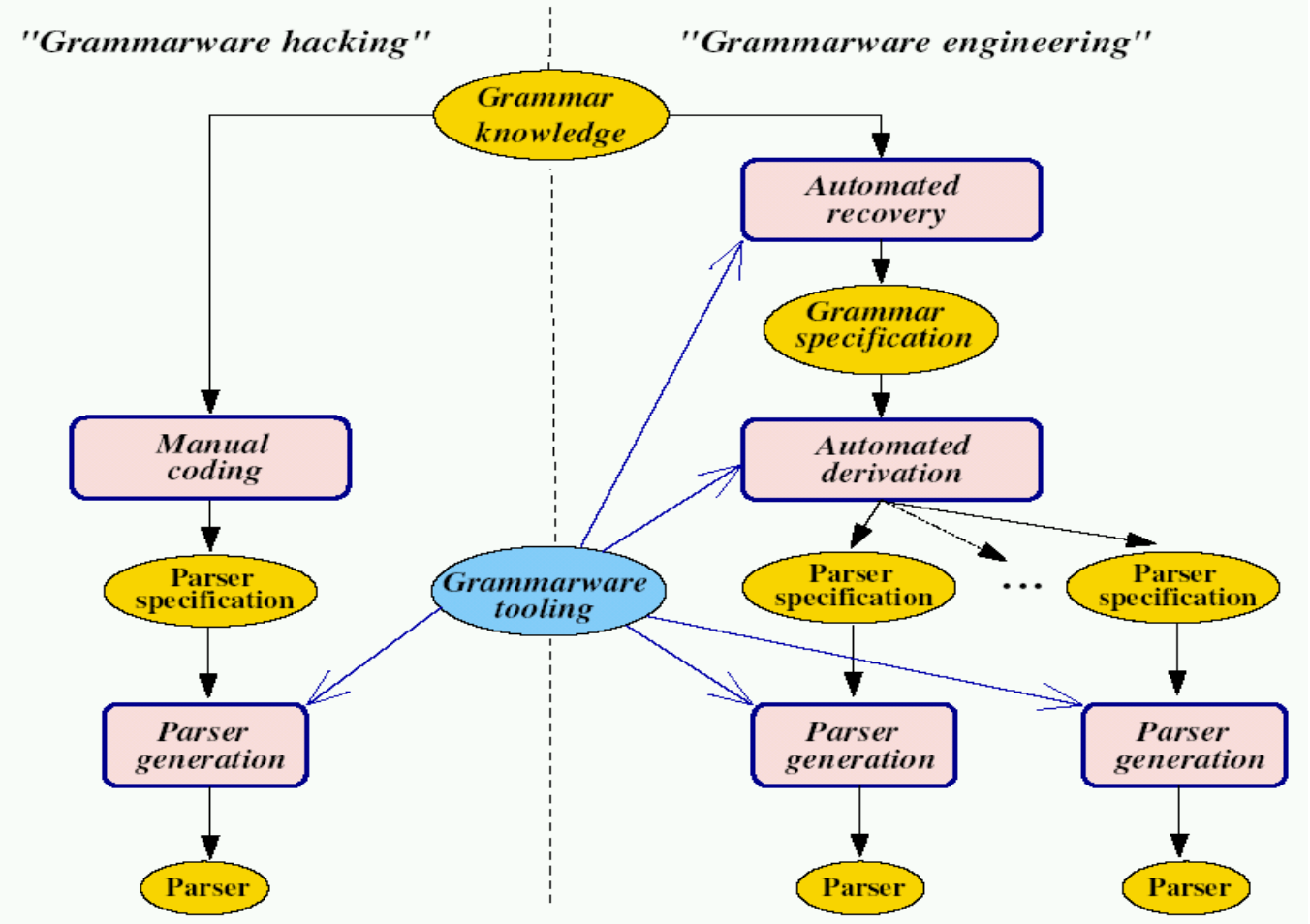


Done

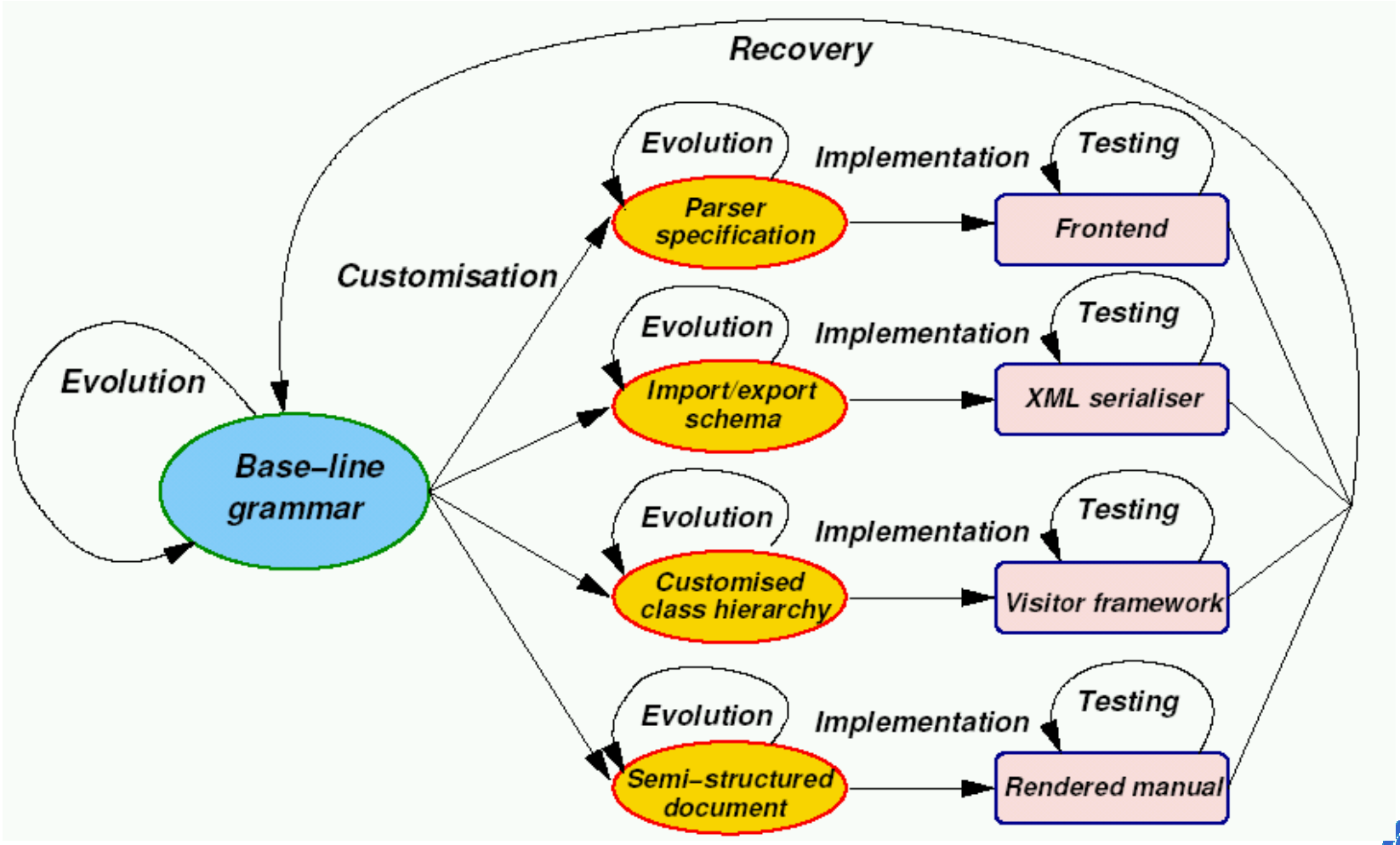
A process for stealing grammars of languages that are used for business-critical applications



Imposing an engineering discipline upon grammars and grammar-dependent software



A life-cycle for grammarware



Next research activity – Aspect-Oriented Cobol

Fact 1: Aspect-Oriented Programming (AOP) is hot and recent.
Fact 2: Error handling is a “crosscutting concern”.
Fact 3: Cobol supports (a form of) AOP since the 1960's.

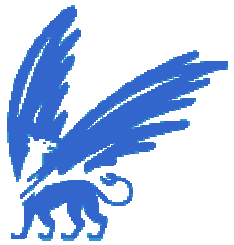
A legacy aspect for file-error handling – matured in 40 years

```
DECLARATIVES.
 HANDLE-F0815-ASPECT SECTION.
  USE AFTER ERROR ON FILE-F0815.
 HANDLE-F0815-ADVICE.
  MOVE "F0815"      TO PANIC-RESOURCE
  MOVE "FILE ERROR" TO PANIC-CATEGORY
  MOVE FILE-STATUS  TO PANIC-CODE
  GO TO PANIC-STOP.
END DECLARATIVES.
```

Joint work with
Kris De Schutter

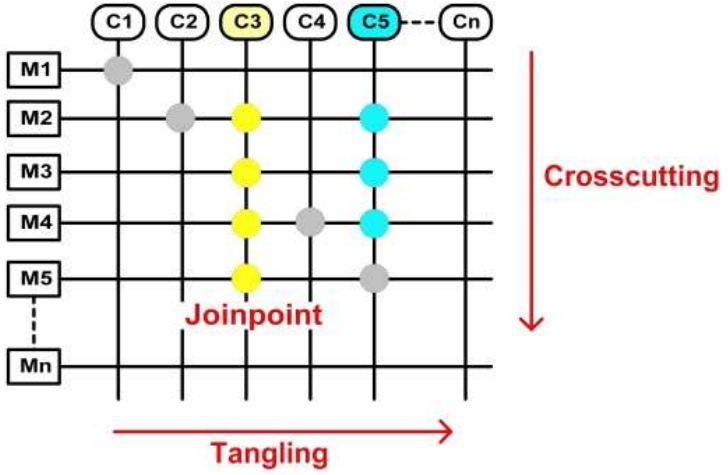
UNIVERSITEIT GENT

vrije Universiteit amsterdam



Aspect-oriented programming basics

What's an aspect?
A ***crosscutting*** concern.



What's aspect-oriented programming?
Its two things:

- ***Obliviousness***: base modules are largely unaware of crosscutting concern.
- ***Quantification***: crosscutting concerns can address a multitude of joinpoints.

Joint work with
Kris De Schutter

UNIVERSITEIT GENT

vrije Universiteit amsterdam

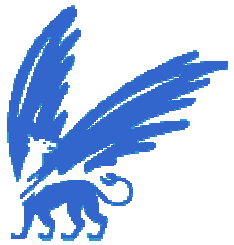
A list of aspects in Cobol or elsewhere

- Logging/Tracing
- Error handling
- Synchronisation
- Persistence
- Security, authorisation
- Design-by-contract
- Protocol enforcement
- ...

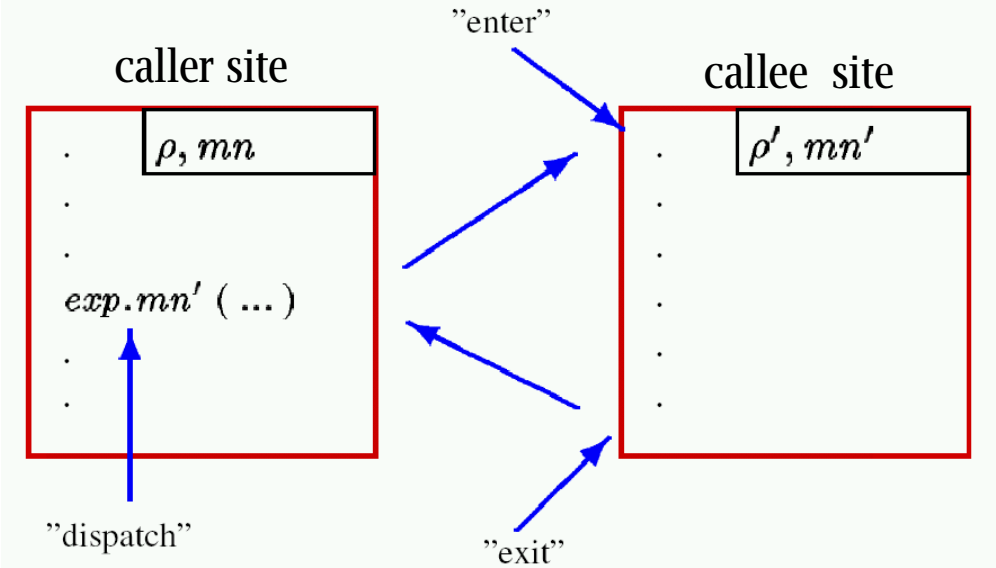
Joint work with
Kris De Schutter

UNIVERSITEIT GENT

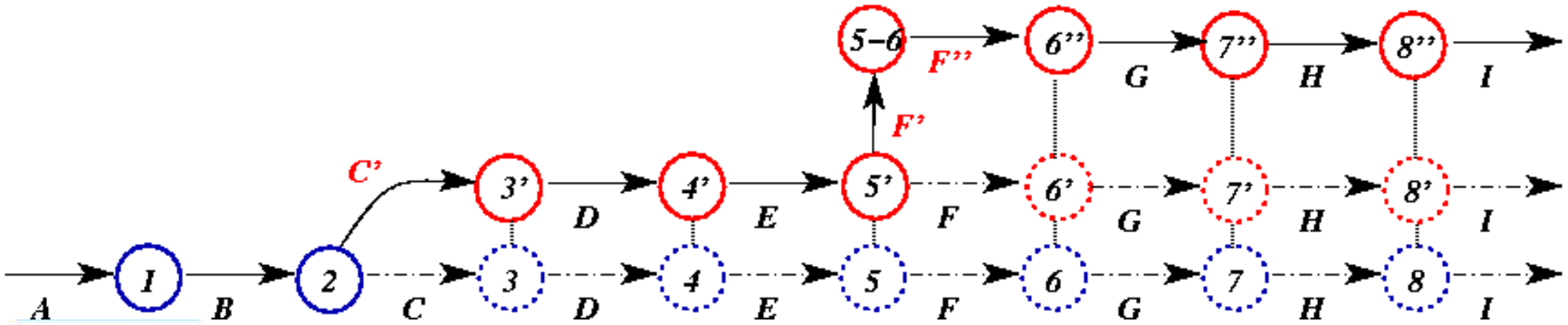
vrije Universiteit amsterdam



Method-call interception



No aspect, one aspect, two aspects in a state transition system



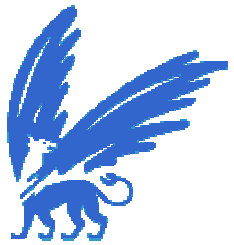
Structure of an intra-program aspect in Cobol

```
IDENTIFICATION DIVISION.
 PROGRAM-ID. <program-id>
…
PROCEDURE DIVISION USING …
 DECLARATIVES.
  USE <when> JOIN-POINT
    <pointcut>
   <paragraph-name>.
    <advice-paragraph>
 END DECLARATIVES.
… *> actual program sections follow
```

Joint work with
Kris De Schutter

UNIVERSITEIT GENT

vrije Universiteit *amsterdam*



Structure of an inter-program aspect in Cobol

```
IDENTIFICATION DIVISION.
 ASPECT-ID. <aspect-id>
...
ENVIRONMENT DIVISION.
... *> environment for each affected program
DATA DIVISION.
... *> data for each affected program
PROCEDURE DIVISION.
 DECLARATIVES.
  USE <when> JOIN-POINT
    <pointcut>
  <paragraph-name>.
    <advice-paragraph>
 END DECLARATIVES.
... *> end of aspect
```

Joint work with
Kris De Schutter

UNIVERSITEIT GENT

vrije Universiteit amsterdam

Sample aspect – Automated file opening

```
IDENTIFICATION DIVISION.
 ASPECT-ID. ASPECTS/AUTOOPEN.
PROCEDURE DIVISION.
 DECLARATIVES.
  USE AROUND JOIN-POINT
    (     MATCHES READ    VERB
      OR MATCHES REWRITE VERB
      OR MATCHES WRITE   VERB
      OR MATCHES DELETE  VERB
      OR MATCHES START   VERB
    )
    AND BIND MY-FILE   TO FILE
    AND BIND MY-STATUS TO FILE-STATUS OF MY-FILE.
  MY-AUTOOPEN-ADVICE.
    PROCEED.  *> Try and execute file access statement!
    *> The value 42 stands for "File not opened".
    *> This use of 42 is not a joke.
    IF MY-STATUS = 42
     OPEN I-O MY-FILE
     PROCEED.  *> Try again!
 END DECLARATIVES.
```

Joint work with Kris De Schutter

UNIVERSITEIT GENT

vrije Universiteit *amsterdam*

Cobol's join-point model

Pointcuts:

- `MATCHES` <verb> `VERB`
- `MATCHES GETTER`
- `MATCHES SETTER`
- `CALL TO` <program-name>
- `BIND` <name> `TO` <subject>
- <pointcut1> `AND` <pointcut2>
- `NOT` <pointcut>
- `TRY` <pointcut>
- `CFLOW` <pointcut>
- `THROWS` <exception>
- ...

Subjects:

- `JOIN-POINT`
- `NAME OF` <subject>
- `TYPE OF` <subject>
- `IDREF OF` <subject>
- `LEVEL 01 OF` <subject>
- `FILE OF` <subject>
- `FILE STATUS OF` <subject>
- ...

"`OF JOIN-POINT`" can be omitted

Joint work with
Kris De Schutter

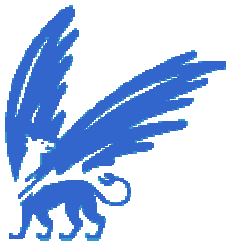
UNIVERSITEIT GENT

vrije Universiteit *amsterdam*

Time to sum up ...

What should computer scientists learn from Cobol?

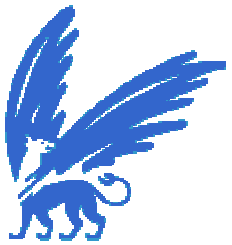
1. Scientists are amazingly unaware of (Cobol's) reality!
2. Let's pay attention to scalability, complexity, heterogeneity!
3. Let's understand IT contributions in a time frame of decades!
4. Cobol comes with several well-designed wheels; don't re-invent!
5. Domain-specific languages do exist! Cobol design is DSL design!



What should computer scientists do for Cobol?

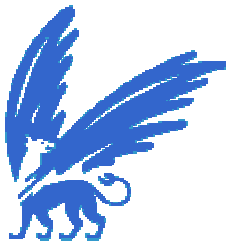
1. Move business programming into the research focus!
2. Pay attention to deployed systems as they exist!
3. Provide suitable courses in the software engineering curriculum!
4. Support work of ISO/ANSI!
5. ***Stop bashing Cobol!***

Let's disagree with Edsger Dijkstra, when he says: "*With respect to COBOL you can really do only one of two things: fight the disease or pretend that it does not exist. Most Computer Science Departments have opted for the latter easy way out.*"



Some research challenges in the Cobol context

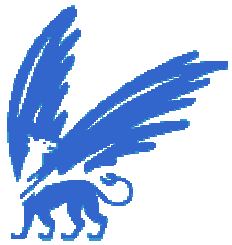
- Parsing and pretty printing
- Automated transformation
- Automated testing
- Design-by-contract
- Code obfuscation
- Business-rule extraction
- User-interface migration
- Data reverse and re-engineering
- Language reference engineering



What must be changed about Cobol?

1. *Not* the acronym.
2. *Not* the language (modulo Cobol's normal evolution).
3. *Not* the application domains (don't mind even more domains).
4. *Not* the standardisation process (modulo grammarware engineering).

So nothing?



Last slide

What has to change is the public understanding of Cobol:

Cobol is more than a matured language. Cobol environments provide an evolving, interoperable solution for business computing problems.

to Wim/W4/J4/... for Cobol

Cobol