Reverse Engineering in practice

Joost Gabriels, Serguei Roubtsov, *Alexander Serebrenik*
LaQuSo & SET

TU/e Technische Universiteit Eindhoven University of Technology

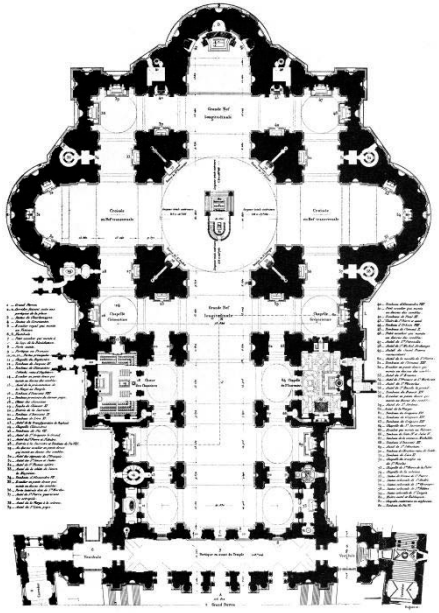
Where innovation starts

Engineering principle

- Construct
- Improve
- Maintain
- Assess
- (Re)use

} objects

We need models!



Models?

- **Model is an object similar to an original object in important features:**
  - **What is important?**
- **Modelling aims at new knowledge about the original by analysing the model.**
- **Conclusion by analogy: If two objects are similar in some features, then they are similar in other (yet unknown) features.**

Types of models

- **Physical** (have the same physical nature as an original):
  - e.g. a model of a plane tested for aerodynamics

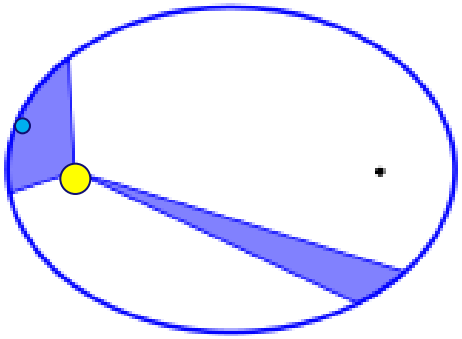
- **Abstract** (may have different physical nature):
  - Textual descriptions

"A line joining a planet and the sun sweeps out equal areas during equal intervals of time ."

  - Mathematical

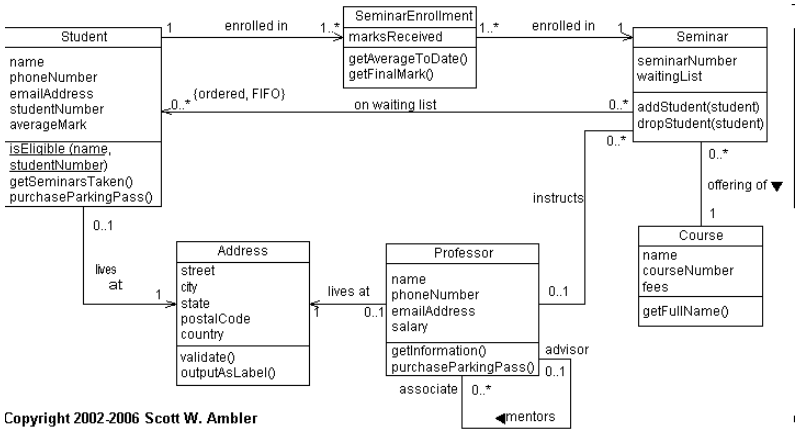
$$\int_{t_1}^{t_2} \frac{1}{2} \cdot base \cdot d(height) = \int_{t_1}^{t_2} \frac{1}{2} \cdot r \cdot r\dot{\nu} dt = \frac{1}{2} \cdot \ell \cdot (t_2 - t_1)$$

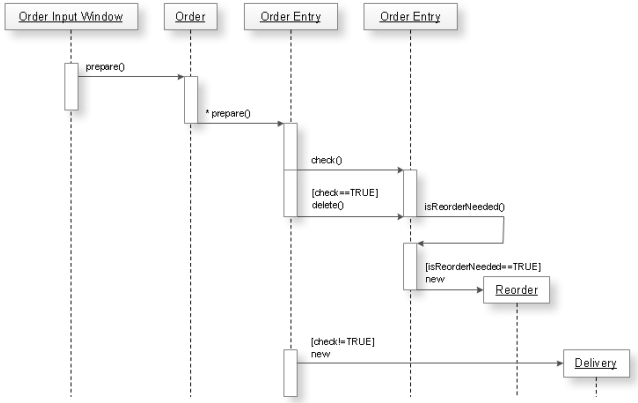
  - Graphical



Software: Models, models and more models

- **Structure**
  - UML deployment diagrams
  - UML class diagrams
  - Dependency graphs
- **Behaviour**
  - UML sequence diagrams
  - UML activity diagrams
  - UML state diagrams
  - (Coloured/timed) Petri nets
  - Process algebras
  - Flow charts
  - Performance model
  - Communication model





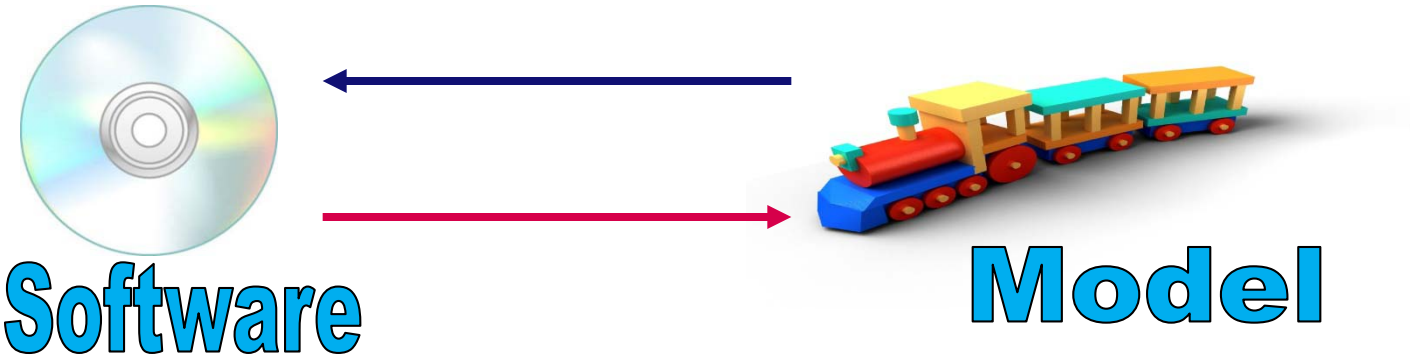
Sounds familiar? Kruchten's 4+1 views

	Static (structure)	Dynamic (behaviour)
Abstract	Logical	Process
Concrete	Development (code in files)	Deployment (processors)

+ Use case scenarios traced through the architecture

Reverse Engineering?

Deriving model from the code.



Ingolf Krüger, U. of California:

"We have been successful in moving from models to code, the challenge is round-trip engineering"

That's what reverse engineering is about!

From Code to Models – Why?

- Consistency check
  - Implementation vs. documentation
- Understanding software
  - In absence of architectural documentation
- Software quality assessment
  - Models may be easier to analyze
  - Software models are often graphical
- Preliminary step for
  - Re-engineering
  - Migration
  - High-level documentation generation

Reverse Engineering Approach

- **Depends on**
  - **What kind of model would we like?**
    - **structure / behaviour**
    - **precise / approximate**
  - **What kind of code do we have?**
    - **complete / incomplete**
    - **compilable / executable / neither**
    - **programming languages: heterogeneous / homogeneous**
    - **“special cases”:**
      - **process models**
      - **business rules**

No silver bullets!

From code to model

Code

Data

Model

Information

Knowledge

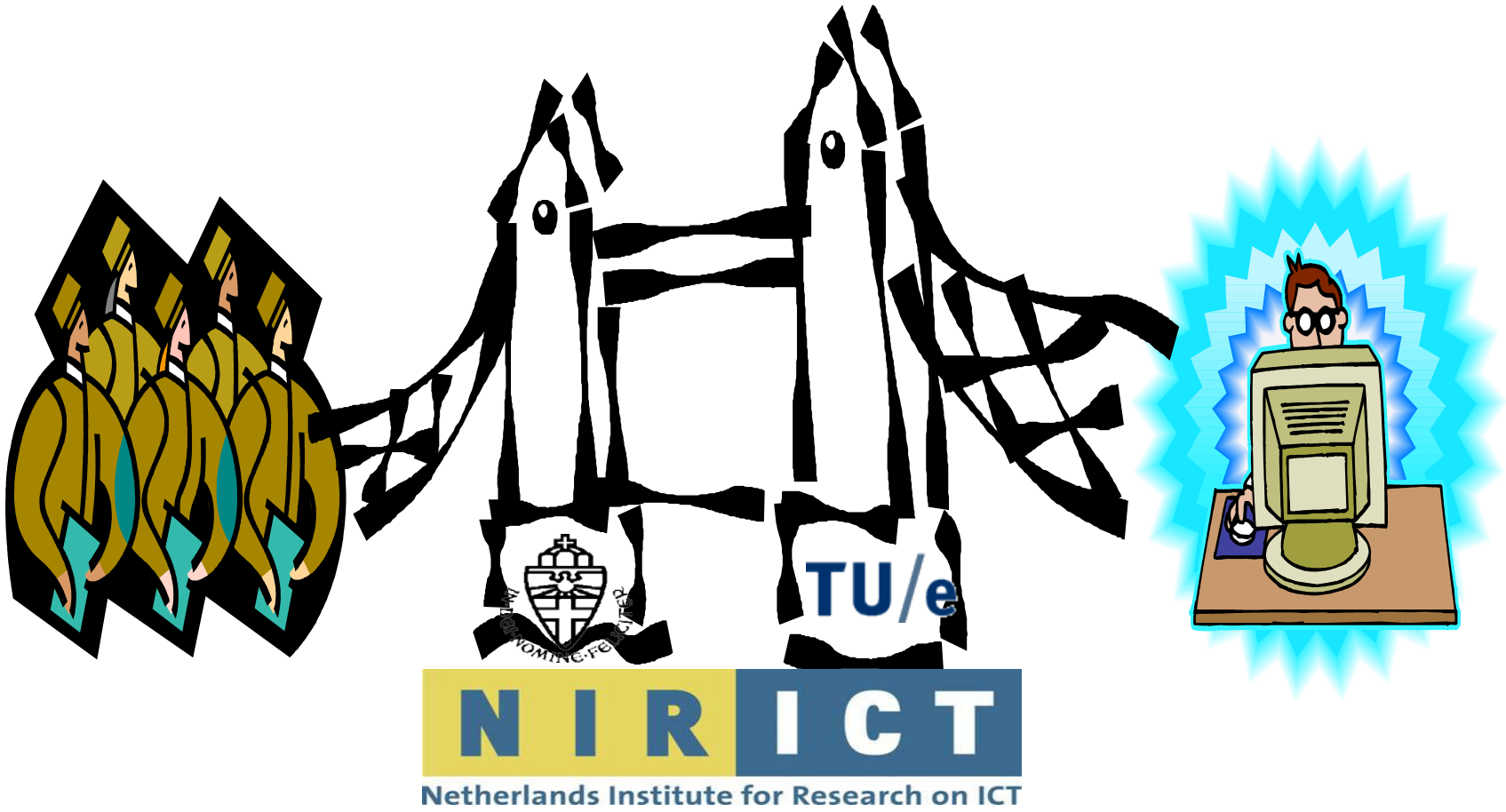
Reverse Engineering Approach

- **Code ⇒ Data**
  - Parsing
  - Scripting
  - Focused search (grep, …)
- **Data ⇒ Model**
  - Fact extraction
- **Model ⇒ Information**
  - Measurement
  - Visualisation
- **Code ⇒ Information**
  - Inspection
  - Walkthrough
- **Information ⇒ Knowledge**:
  - Review

Case studies

1000 facets of reverse engineering based on LaQuSo case studies

LaQuSo?



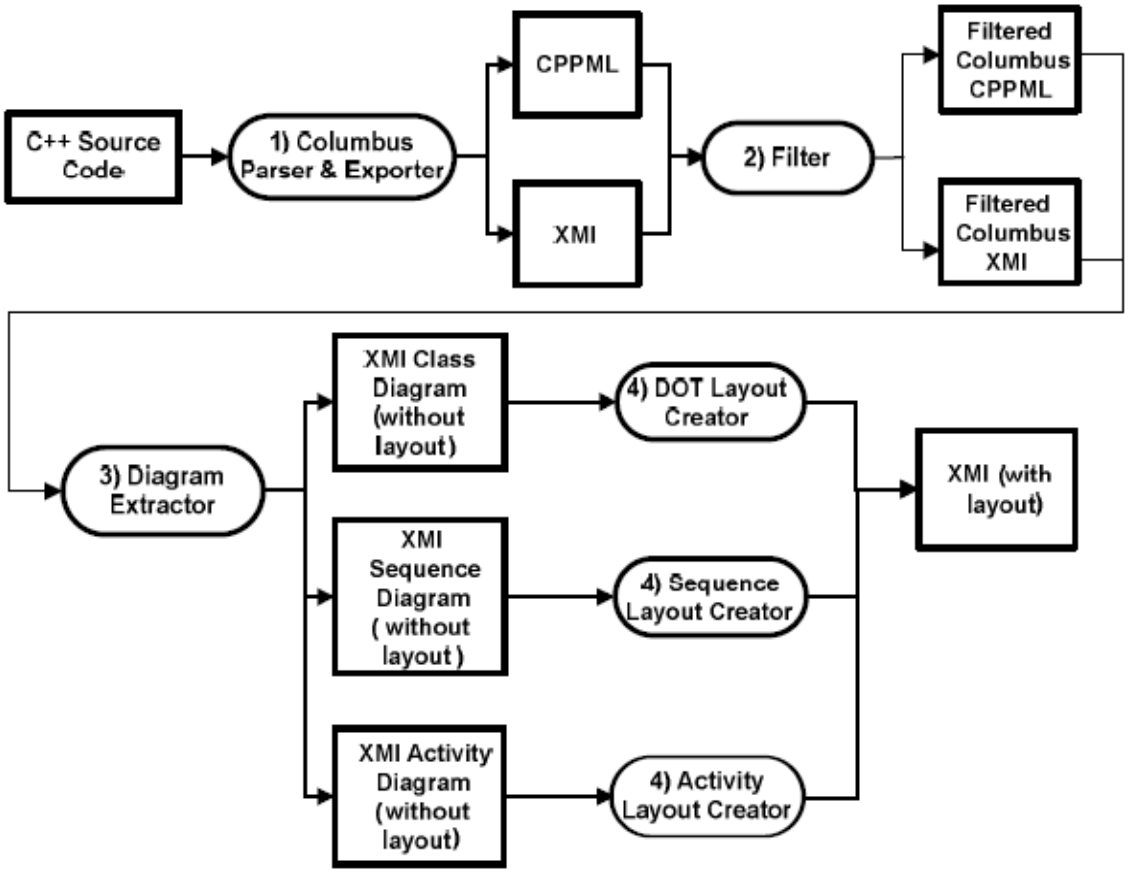
Toy example: Bellflower

- What kind of code do we have?
  - complete
  - compilable
  - programming languages: homogeneous: MS VS C++
  - originates from Rational Rose models
    - Original models vs. inferred models!

- What kind of model would we like?
  - structure: UML class diagrams
  - behaviour: UML sequence diagrams
  - precise

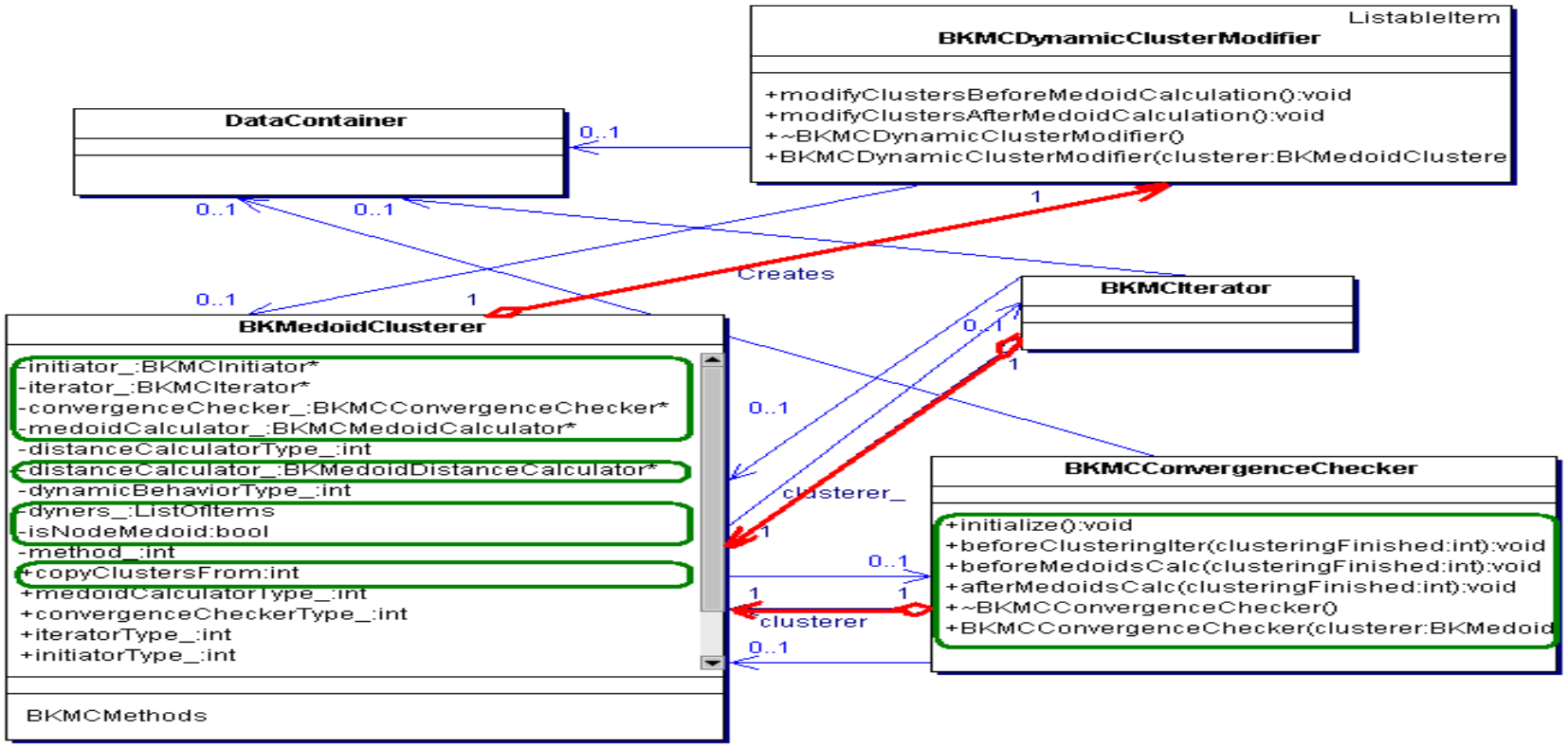
What did we do? Approach

- Code ⇒ Data
  - Parsing
- Data ⇒ Model
  - Fact extraction:
    - Filtering
    - Diagram extraction
- Model ⇒ Information
  - Visualisation
- Information ⇒ Knowledge
  - Review



CPP2XMI

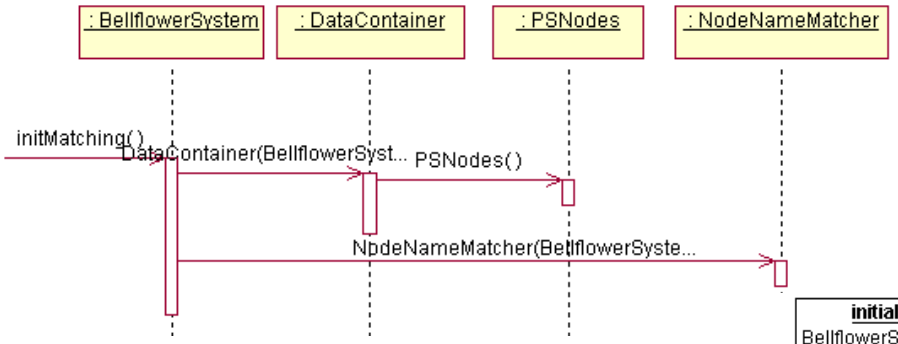
Information (structure)

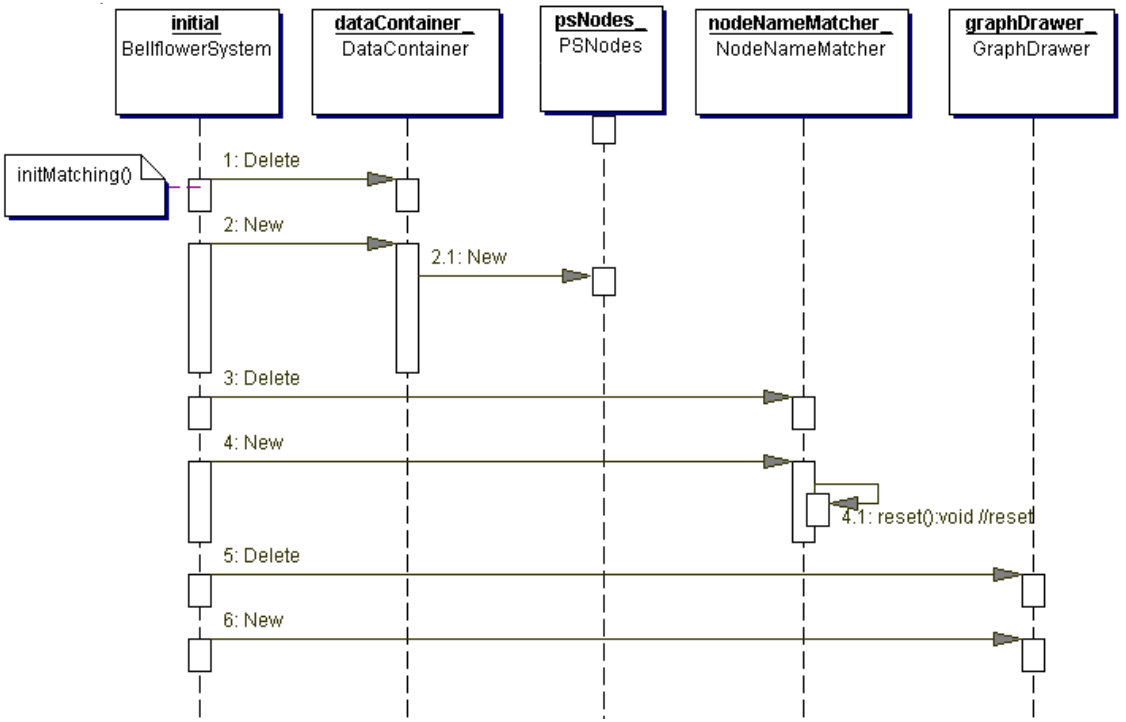


Inferred class diagram contains more details than the original one:

- **Additional fields and methods in certain classes**
- **Additional relationship: aggregation**

Information (behaviour)





Sequence diagrams:

- **The inferred one contains more detailed behaviour: new() and delete() methods and**
- **One more object derived from implementation**

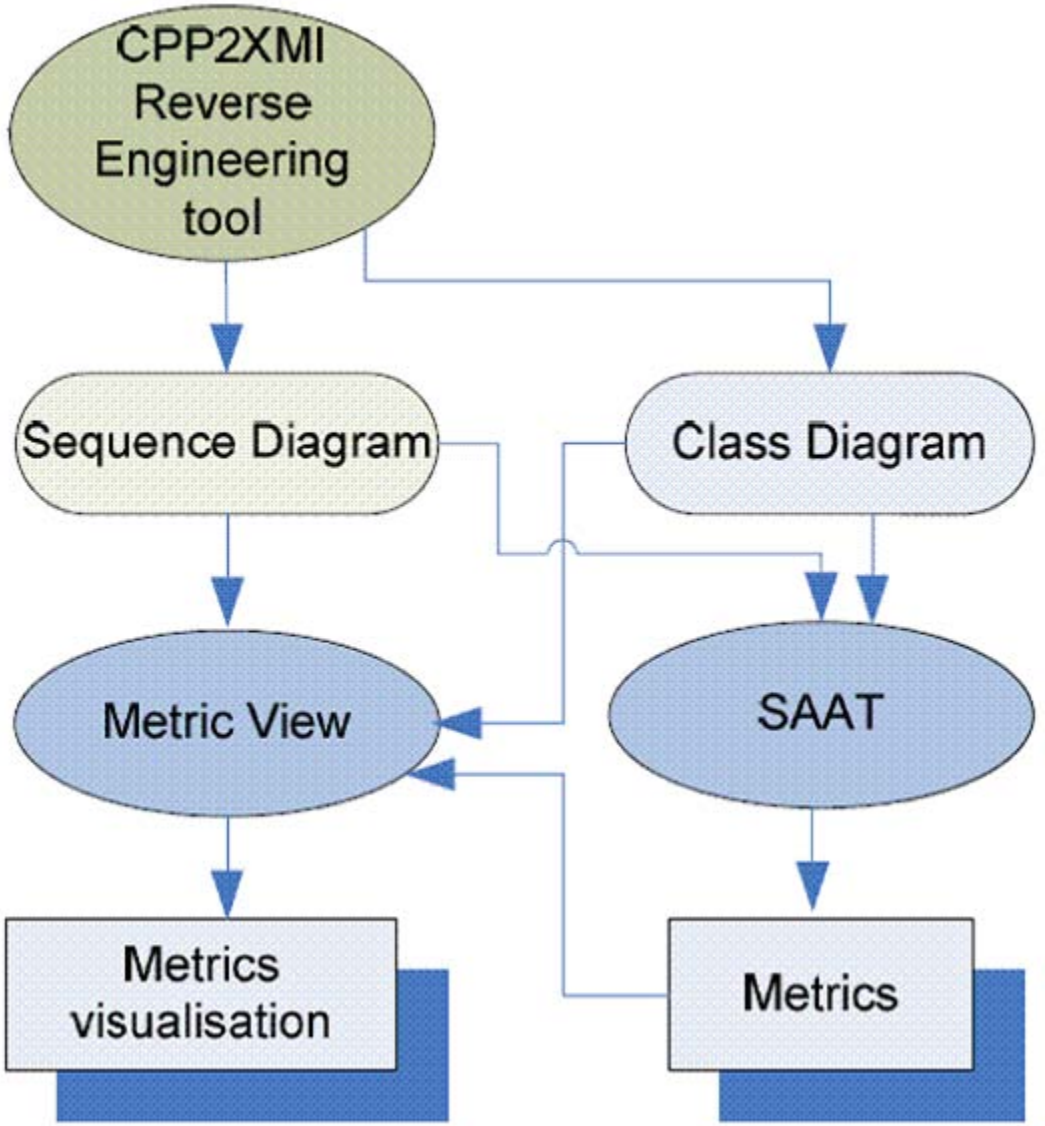
Inferred model is consistent wrt design.

Industrial case with CPP2XMI: Printer-producing company's software

- What kind of code do we have?
  - complete
  - compilable
  - programming languages: homogeneous: C++
  - 60 KLOC
  - No documentation
- What kind of model would we like?
  - structure: UML class diagrams
  - behaviour: UML sequence diagrams
  - precise

What did we do? Approach (continued)

- **Model ⇒ Information**
  - **Measurement**
  - **Visualisation**
- **Information ⇒ Knowledge: Review**



1 picture = 1000 words?

unused classes

We need something better...

Metrics (1)

Metrics	Subsystems	
	A	B
Number of classes	176	70
Number of methods	1106	383
Avg. methods per class	6.28	5.45
Classes with > 30 methods	4	2
Max fan-in / Max fan-out	27 / 27	23 / 21

- Subsystem A is quite big.
- Big parts of functionality are implemented in a few files.
- Many files depend on these few.

Metrics (2)

- **Models derived: (illegible) sequence diagrams**

Metrics		Subsystems	
		A	B
Incoming and outgoing messages per class	Maximum	112	271
	Classes with > 30 mess.	5	6
Max. depth of scenario		41	55

- **A number of heavily used classes**
- **Scenarios' depth: too high → functionality should be differently distributed.**

So far…

- **Case studies:**
  - **Code**
    - **Complete, compilable, homogeneous (OO)**
  - **Model**
    - **UML class/sequence/activity**
    - **precise**
- **Approach**
  - **parsing, fact extraction, visualisation, measurement**
- **Results**
  - **Precision ⇒ Illegibility (too many details)**
    - **Metrics can be of great help!**

Real life industrial systems

They are often:

- **Not only OO (legacy systems)**
- **Heterogeneous (C/Assembler, Cobol/PL SQL,…)**
- **Incomplete (some code is in libraries and third-party components)**
- **Not compilable and executable within analysis environment (‘weird’ OS, proprietary development environment, …)**

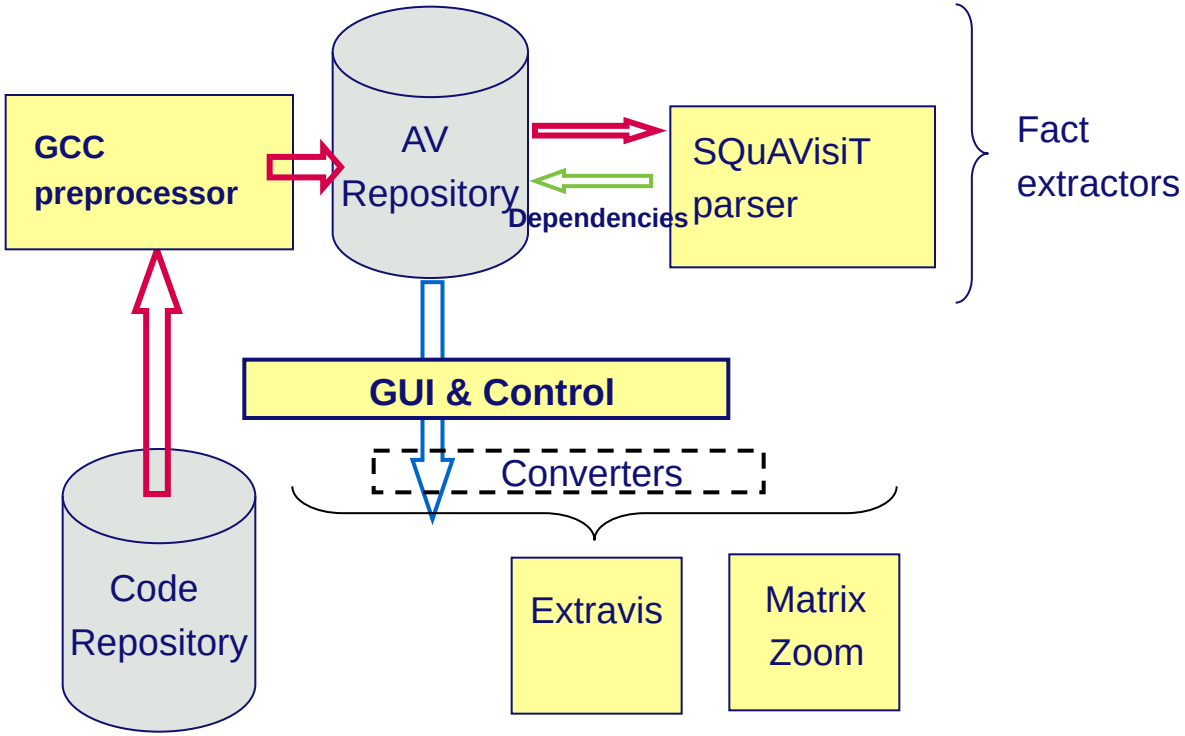
Industrial case with SQuAVisiT: Embedded System

- What kind of code do we have?
  - Not OO
  - Almost homogeneous: mostly C with embedded Assembler
  - Complete
  - Modules of interest are compilable (at least can be parsed)
  - Medium size: 150 KLOC
  - No documentation

- What kind of model would we like?
  - Structure: dependencies and layering
  - Approximate (function pointer calls and Assembler code ignored)

What did we do?

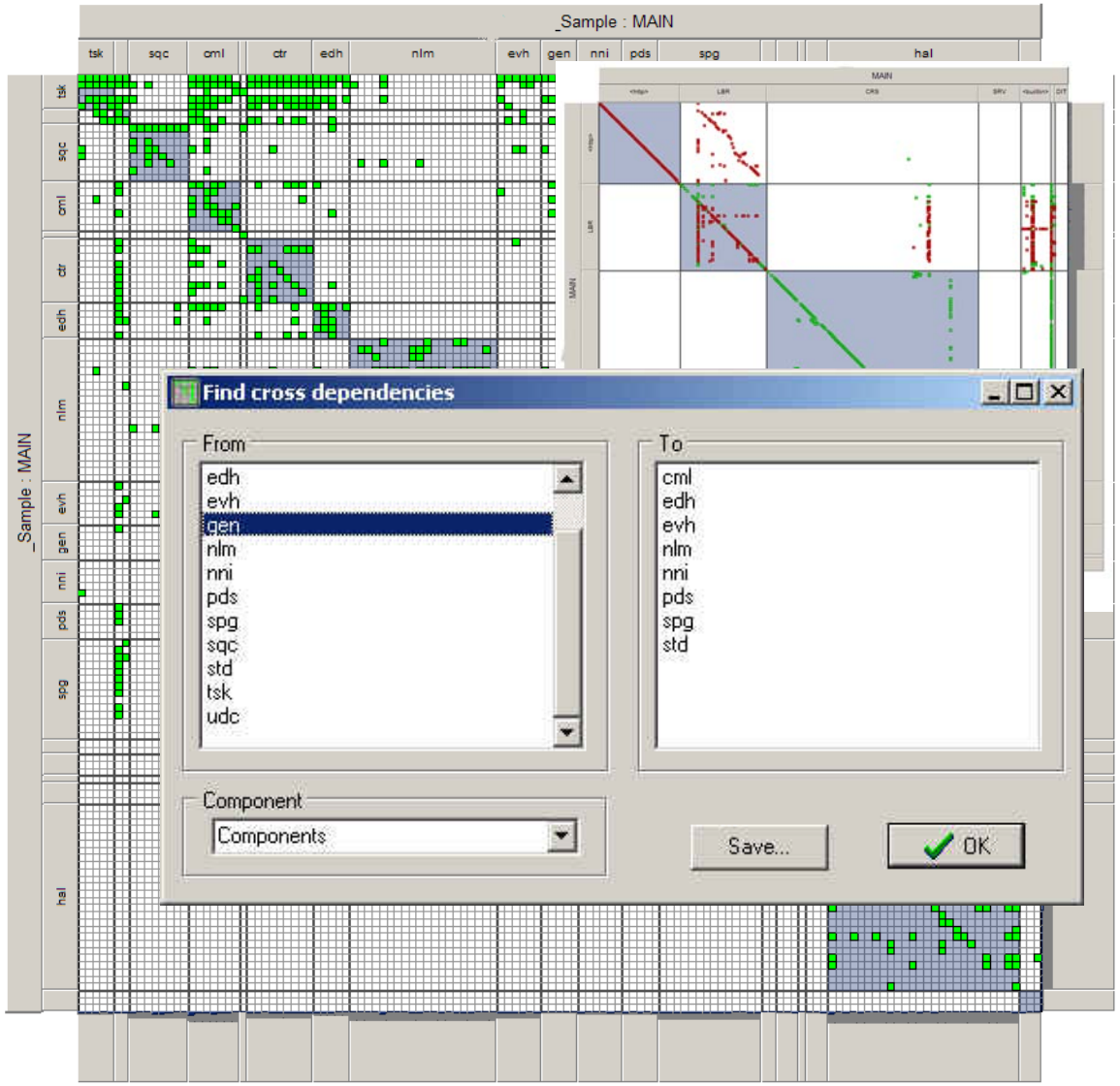
- Code ⇒ Data
  - parsing
- Data ⇒ Model
  - Fact extraction:
    - Dependencies extraction
- Model ⇒ Information
  - Visualisation
- Information ⇒ Knowledge
  - Review



SQuAVisiT

Structure (Matrix View, 2)

- system is poorly layered
- unexpected cross-dependencies exist between components



Case conclusions

- **Non OO systems demand different models:**
  - **Dependencies**

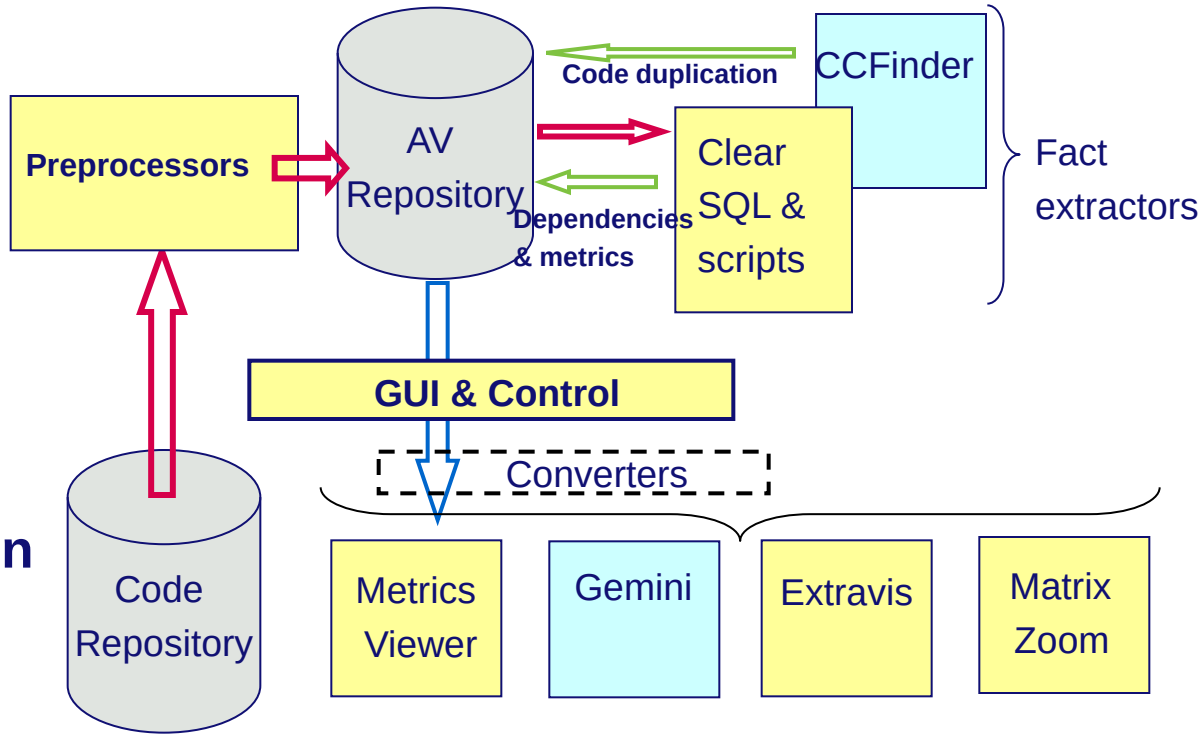
- **Visualisation – key to understanding**

Industrial case: Insurance company's expert system

- What kind of code do we have?
  - Not OO
  - Heterogeneous: Javascript, PL SQL, C++, Java, Cobol
  - Complete
  - Not compilable (less relevant here)
  - Medium size: 300 KLOC
  - Scarce documentation
- What kind of model would we like?
  - Structure: dependencies and layering with implications for maintenance

What did we do? Alternative approach

- Code ⇒ Data
  - Ad-hoc scripting
  - ClearSQL tool
- Data ⇒ ModelS
  - Fact extraction:
    - Filtering
    - Dependency extraction
    - Duplication extraction
- ModelS ⇒ Information
  - Visualisation
  - Measurement
- Information ⇒ Knowledge
  - Review



SQuAVisiT

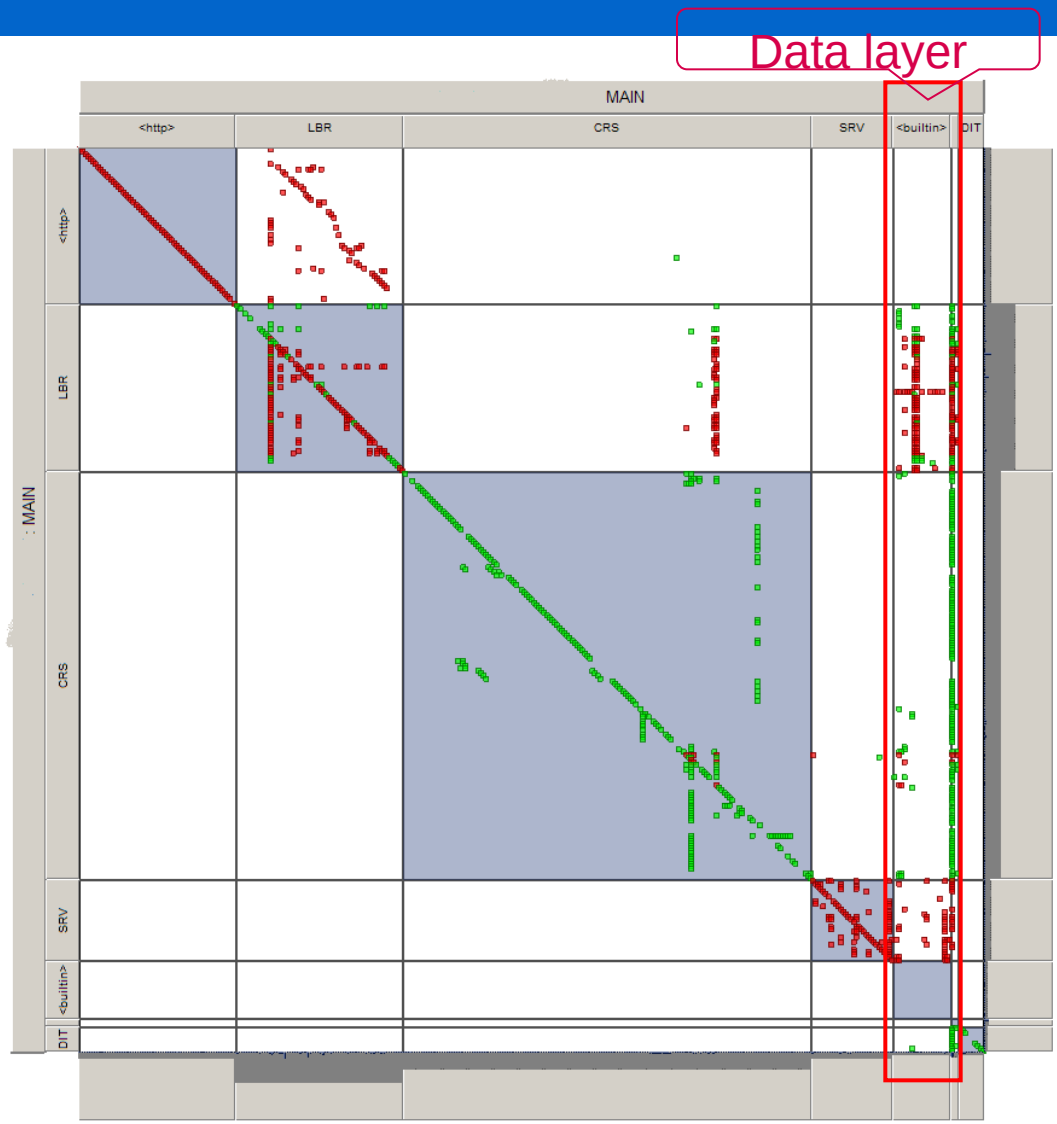
Dependencies Model: Matrix View (1)

Model ⇒ Information

- **(Almost) layered: good design**
  - **BUT data layer is accessed from several layers**

- **Layers affected by calls from top layer are visible (red squares)**

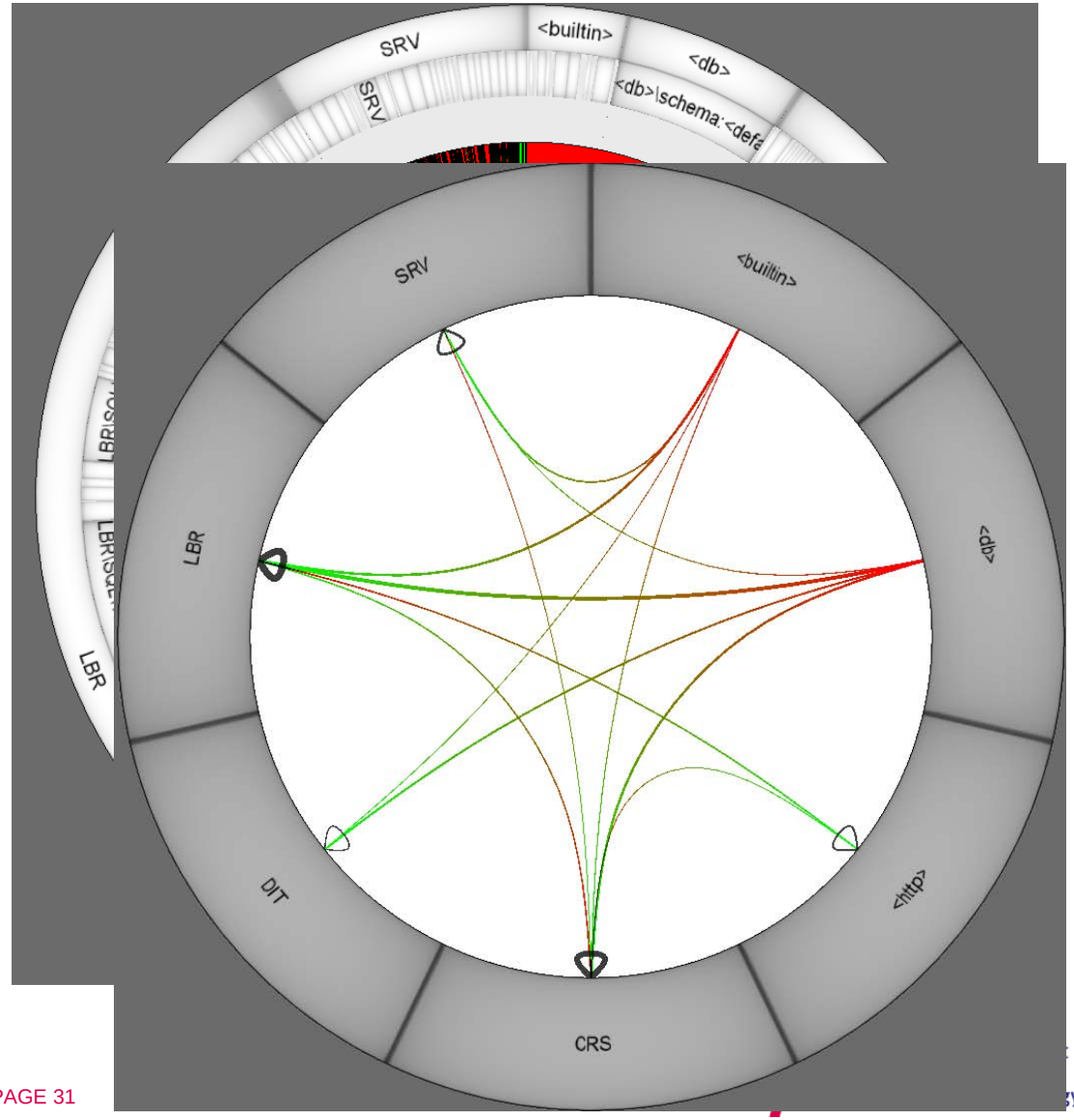
What are the maintenance implications of this figure?



Dependencies Model: Extravis

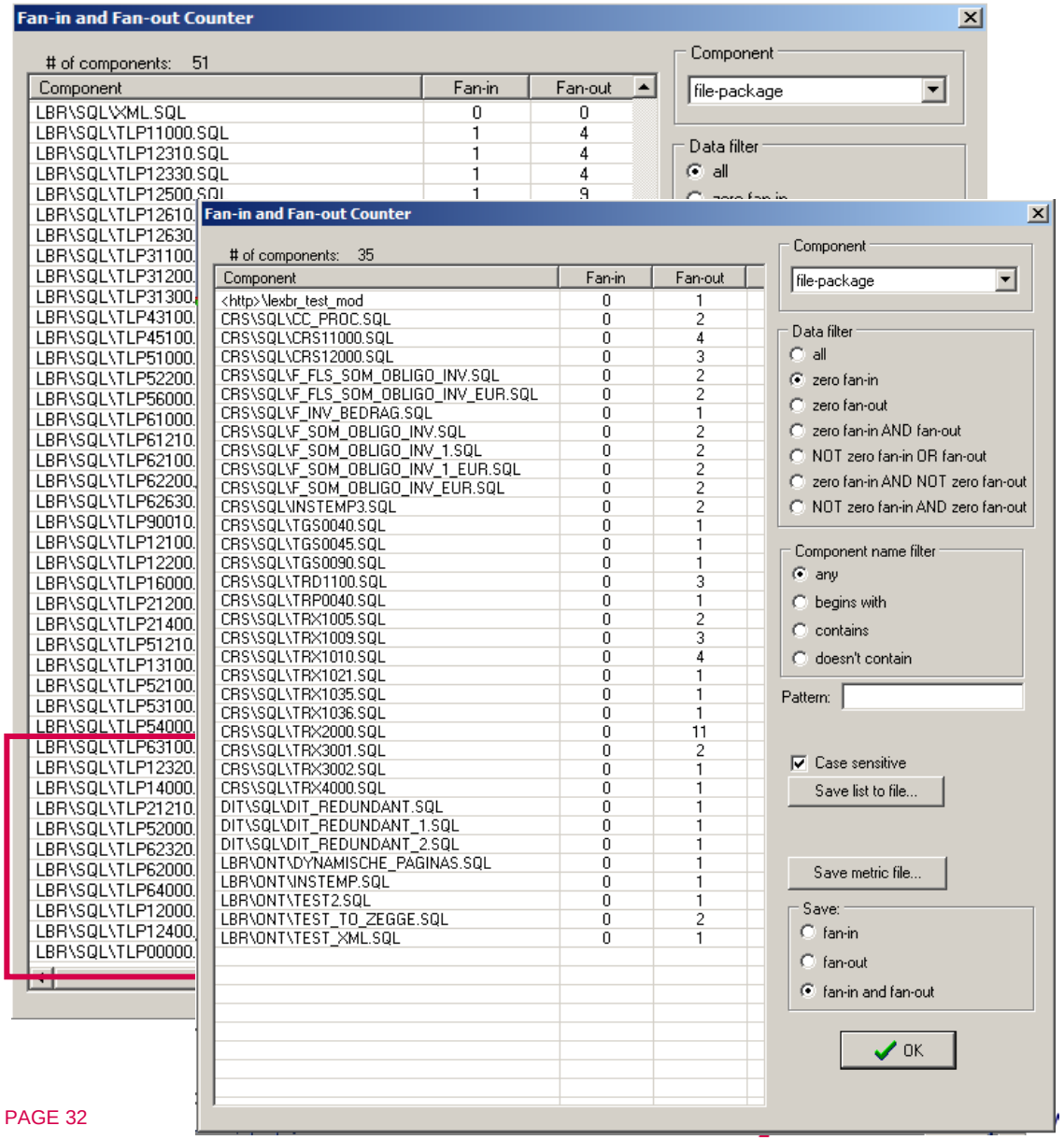
- **Huge green ‘bubbles’ reflect some controversial coding approach: getting rid of parameters by means of naming like f(1,3) -> f_1_3**

- **Absence of dedicated data access layer is confirmed**



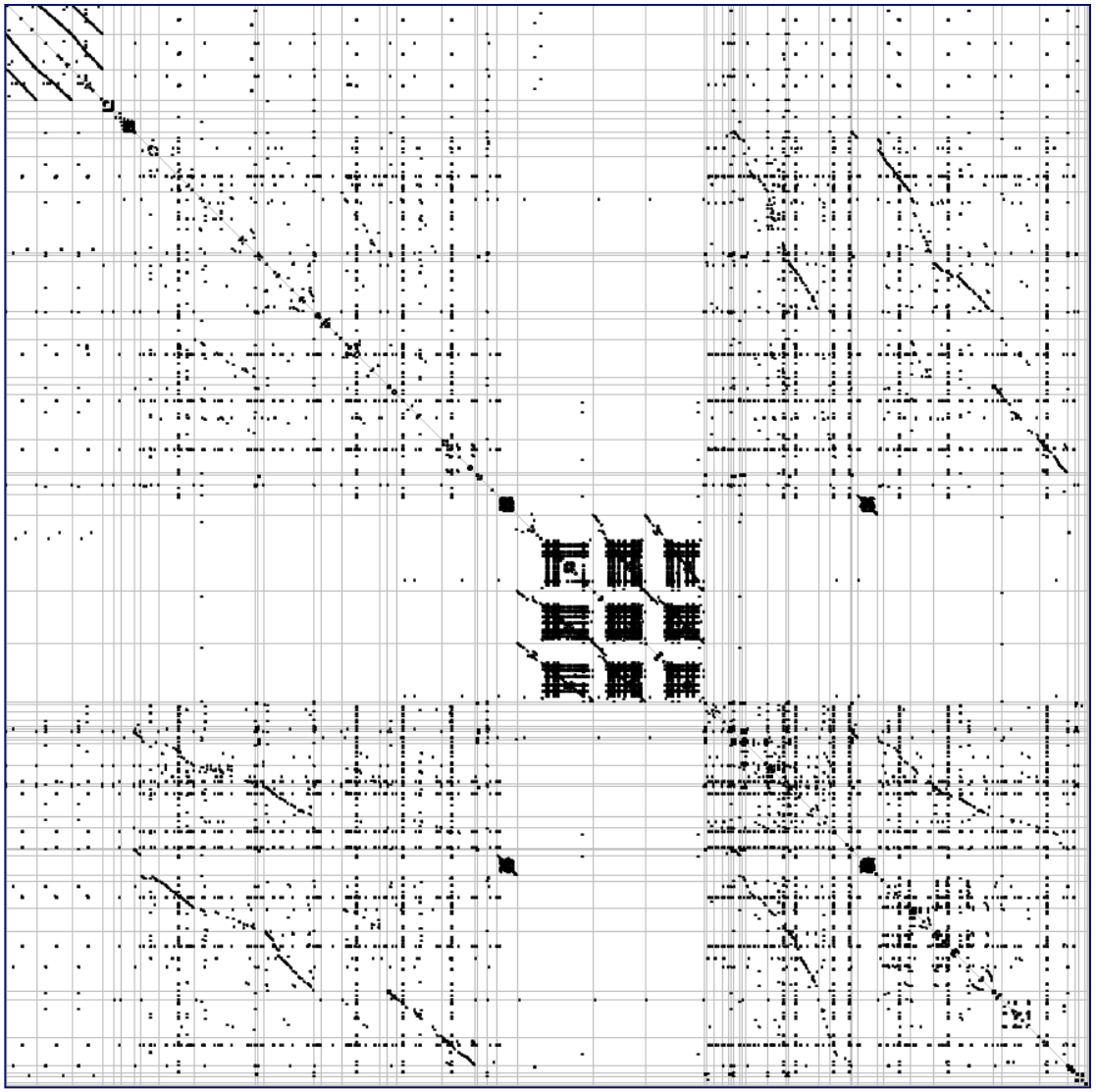
Dependencies Model: Metrics

- ‘Change propagators’ - modules with big Fan-in & Fan-out –bottlenecks
- Modules with zero fan-in – dead code?

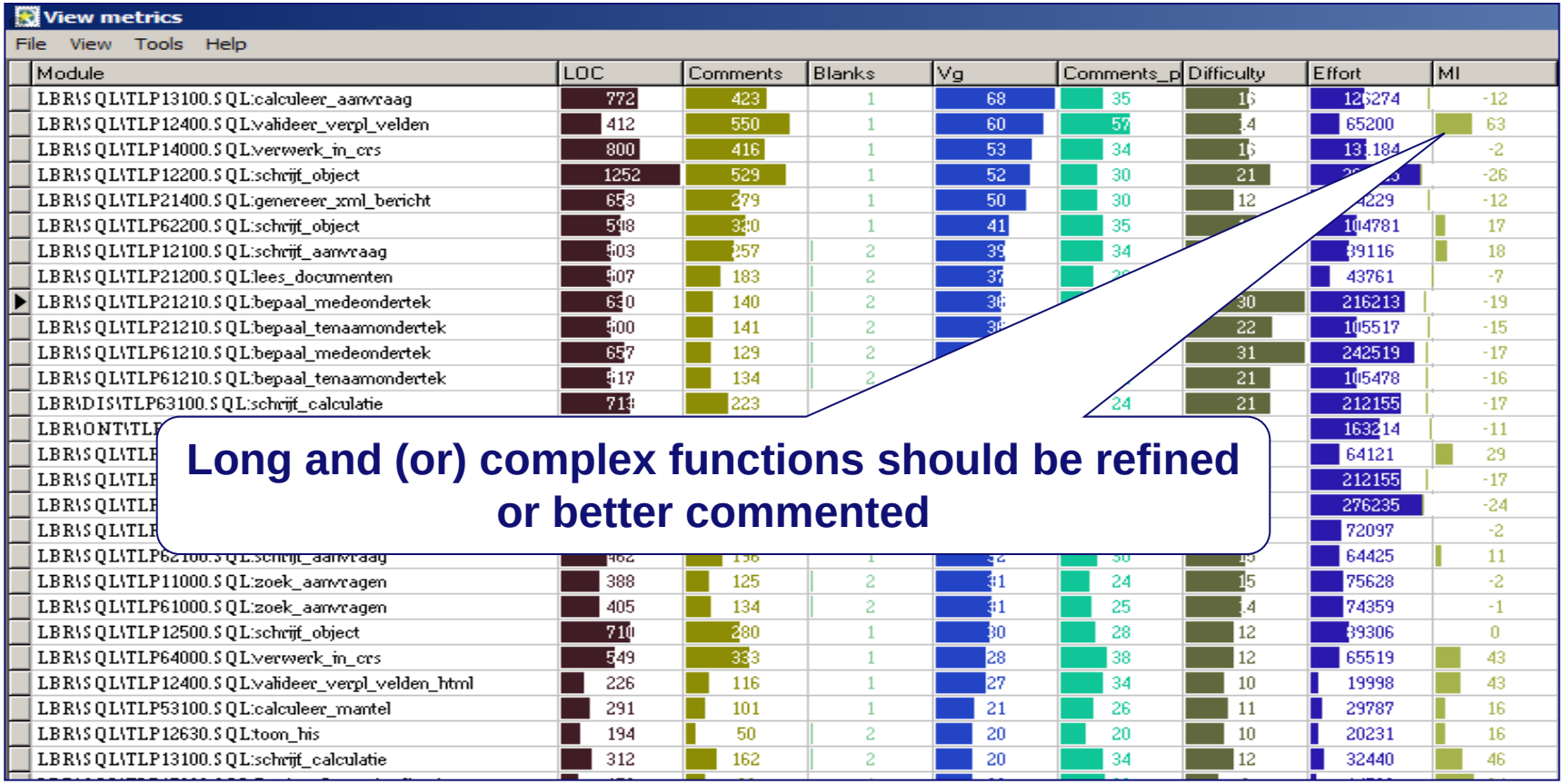


Code duplication model: CCFinder/Gemini

- **Code is polluted with duplication: restructuring would improve maintainability but may change the architecture**



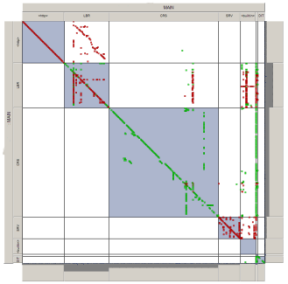
Quality model: Metrics

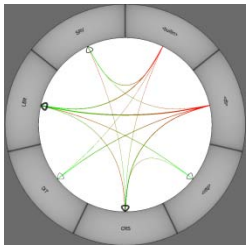


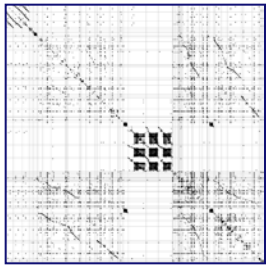
$$MI = 171 - 5.2 * \ln(aveV) - 0.23 * aveV(g') - 16.2 * \ln(aveLOC) + 50 * \sin(\sqrt{2.4 * perCM})$$

Case conclusions

- **Analysis required multiple models:**
  - **Dependency model**
  - **Code duplication model**
  - **Quality model**







What about behaviour? Performance issues with pension fund's 'Calculation engine'

- What kind of code do we have?
  - complete
  - not compilable in analysis environment but executable at the customer's site
  - heterogeneous: PL SQL, Cobol with SQL*Plus inside
  - large size: ~3000 KLOC of Cobol code only
  - abundant sources: Cobol traces, Oracle logs

- What kind of model would we like?
  - Behavioural to explain why the system is so slow

What did we do?

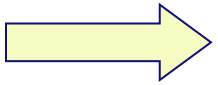
- Running system ⇒ Data
  - Scripting & Focused search in Oracle logs
  - Cobol code instrumentation to obtain traces
- Data ⇒ ModelS
  - Fact extraction:
    - Filtering
- Model ⇒ Information
  - Testing
  - Visualisation
- Information ⇒ Knowledge
  - Code review & visualization analysis

Running system ⇒ Data

Environment:
Cobol application and Oracle DBMS run on the same machine under AIX OS

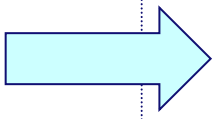
UNIX program 'time' was used to determine execution time for Cobol side

Cobol time



Cobol 'system' time

```
EXEC #5:c=0,e=0,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=3,tim=1112568305
FETCH #5:c=0,e=0,p=0,cr=2,cu=0,mis=0,r=1,dep=0,og=3,tim=1112568305
EXEC #153:c=0,e=0,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=3,tim=1112568305
FETCH #153:c=0,e=0,p=0,cr=2,cu=0,mis=0,r=1,dep=0,og=3,tim=1112568305
FETCH #153:c=0,e=0,p=0,cr=1,cu=0,mis=0,r=0,dep=0,og=3,tim=1112568305
EXEC #179:c=0,e=0,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=3,tim=1112568305
FETCH #179:c=0,e=0,p=0,cr=5,cu=0,mis=0,r=0,dep=0,og=3,tim=1112568306
EXEC #181:c=0,e=0,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=3,tim=1112568306
FETCH #181:c=0,e=0,p=0,cr=3,cu=0,mis=0,r=0,dep=0,og=3,tim=1112568306
EXEC #113:c=0,e=0,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=3,tim=1112568306
FETCH #113:c=0,e=0,p=0,cr=4,cu=0,mis=0,r=1,dep=0,og=3,tim=1112568306
EXEC #201:c=0,e=0,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=3,tim=1112568306
FETCH #201:c=0,e=0,p=0,cr=4,cu=0,mis=0,r=1,dep=0,og=3,tim=1112568306
EXEC #5:c=0,e=0,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=3,tim=1112568306
FETCH #5:c=0,e=0,p=0,cr=2,cu=0,mis=0,r=1,dep=0,og=3,tim=1112568306
EXEC #6:c=0,e=0,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=3,tim=1112568306
FETCH #6:c=0,e=0,p=0,cr=4,cu=0,mis=0,r=1,dep=0,og=3,tim=1112568306
EXEC #7:c=0,e=0,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=3,tim=1112568306
```

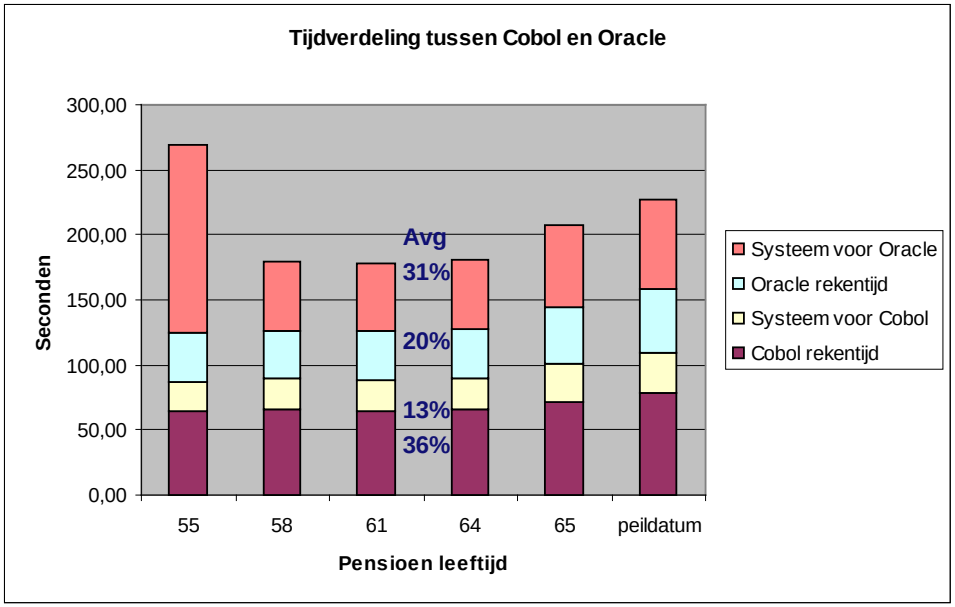


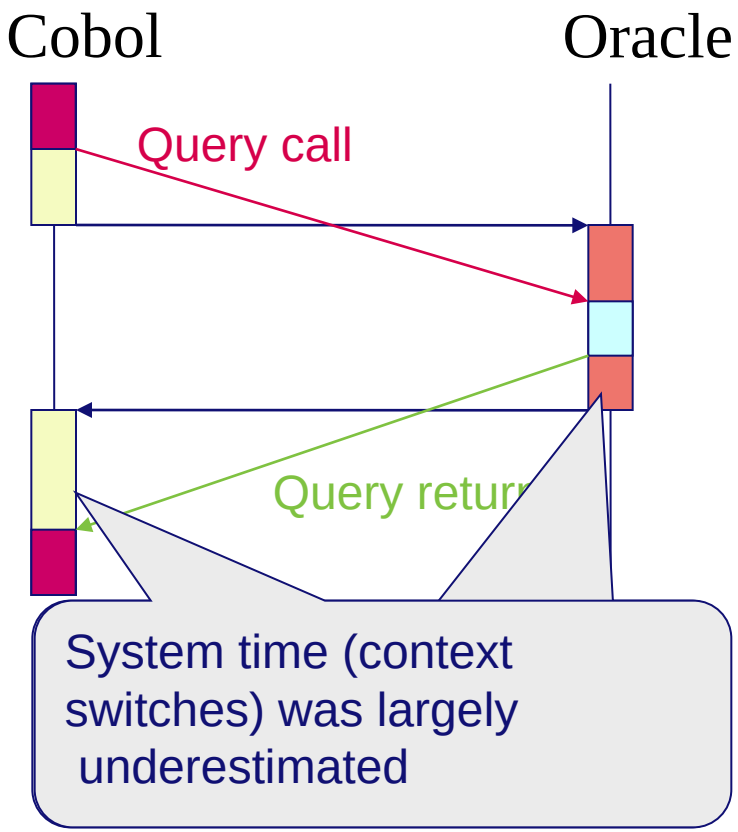
Oracle time

Data ⇒ Communication model

Oracle logs show very moderate execution time. Where is the rest?

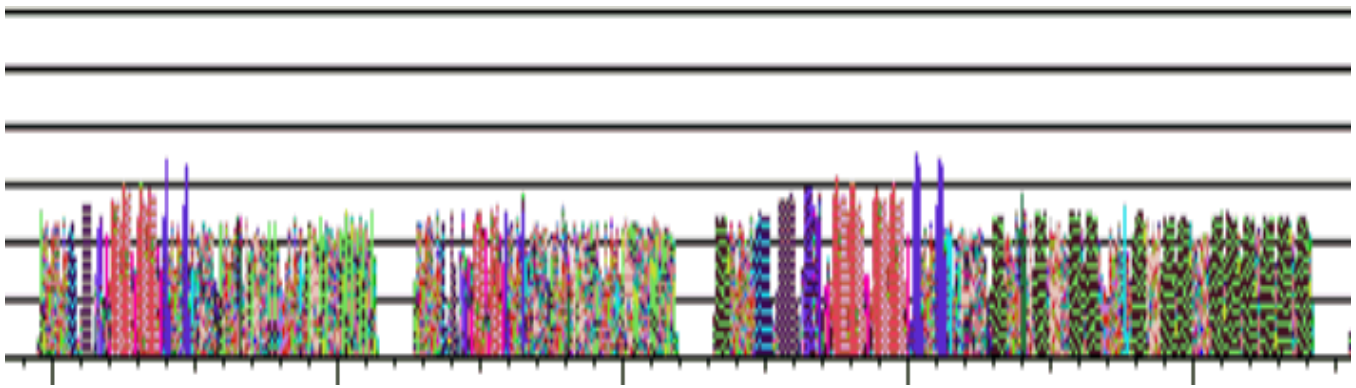
1. Cobol time: pension calculation, query parameters/return processing
2. System time for Cobol: Communication with Oracle, disk usage
3. System time for Oracle: communication with Cobol, disk usage
4. Oracle time for the Cobol application: search in the database





Data ⇒ Performance model

Which queries are most time consuming ?



100 ms: Oracle logging sensitivity

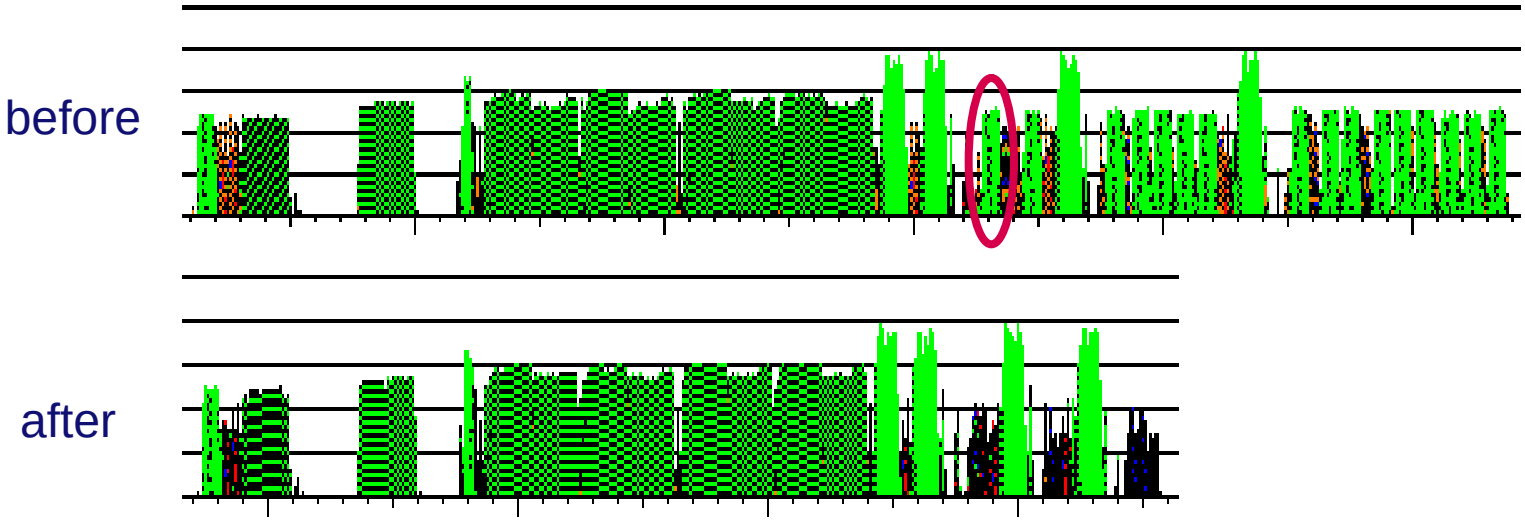
Different colours – different types of queries

- Types can be chosen on the fly
- Adaptable model (or a class of models)

Information ⇒ Knowledge

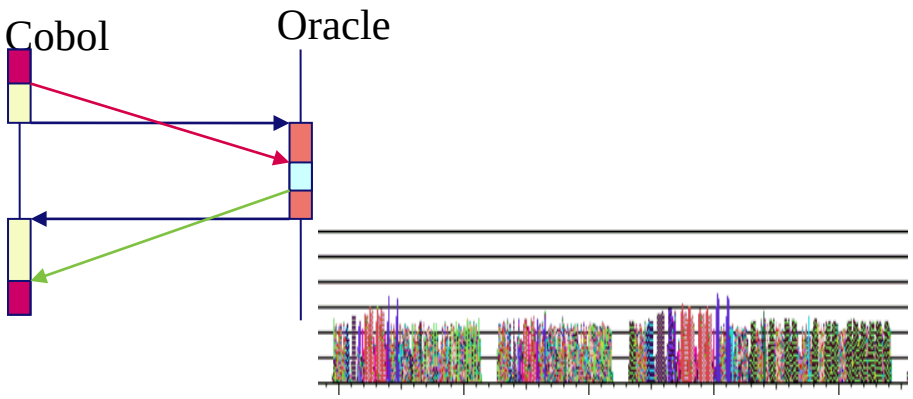
- **Get rid of parameter ‘up-to-datedness’ control**

- **Make use of Cobol ‘static memory’**



Case conclusions

- **Behavioural model**
- **Data is obtained**
  - **From a running system**
  - **By different means**
- **Multiple models**
  - **Communication**
  - **Performance**

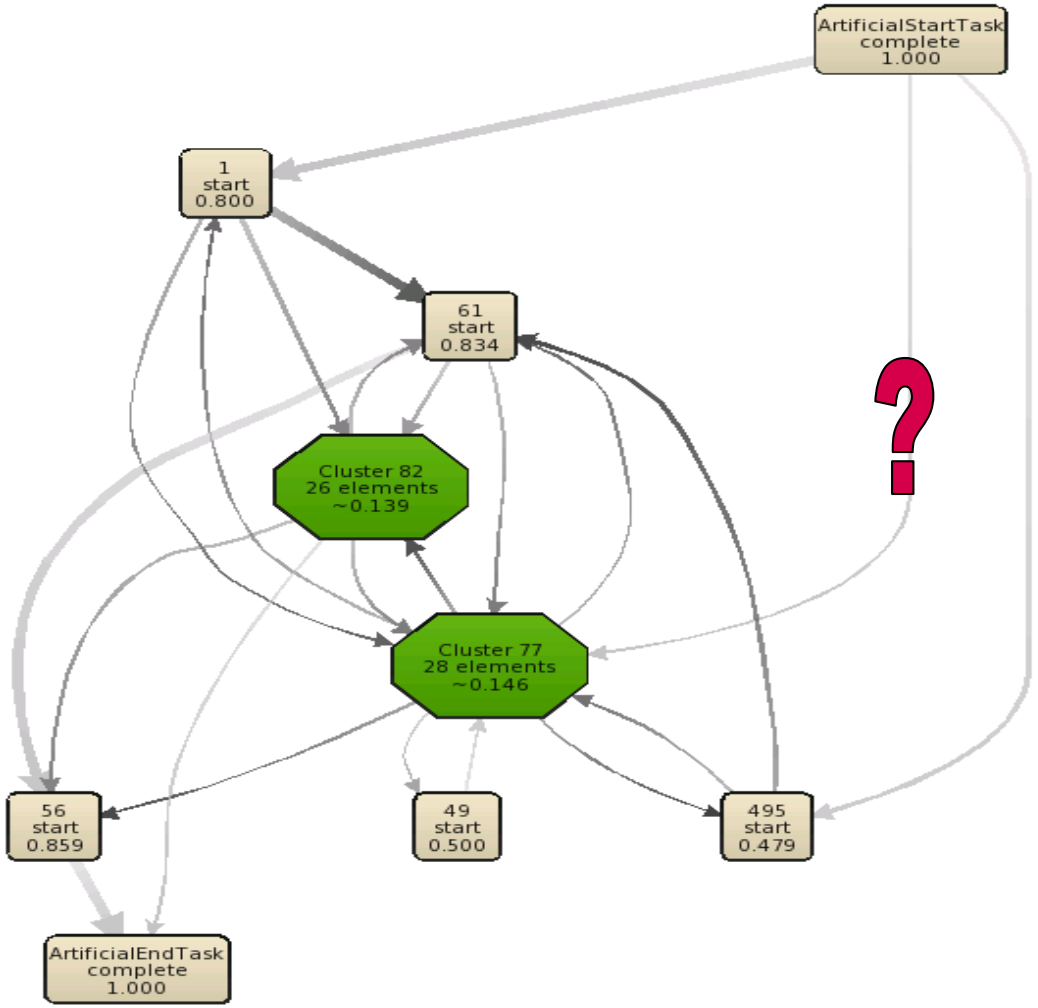


Industrial case: Certificate issuing

- What kind of code do we have?
  - Workflow system log
  - Context:
    - Certificates are requested
    - Data is analysed
    - Certificates are granted (or not)

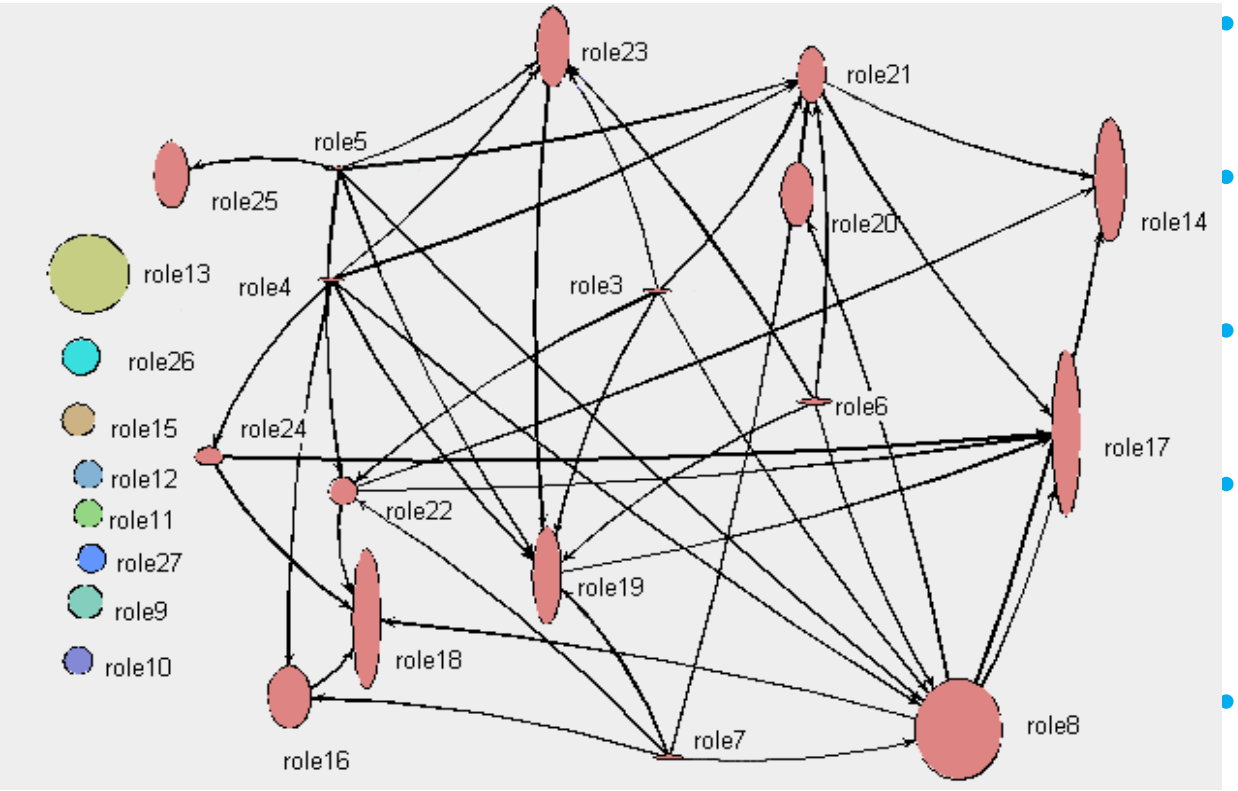
- What kind of models would we like?
  - Process models: granted certificates only
  - Task transfer model
  - Performance model

Process Model



- **Thickness of arrows shows frequency**
  - **Thin lines = anomalies (?)**
- **Closely interrelated tasks are clustered**

Task transfer



- Height: incoming arcs
- Width: outgoing arcs
- 3-7: flat
  - Process initiators
- 14: tall
  - Process finalisers
- 13, 26: disconnected
  - Incidental participants
- 8: many in/out-arcs
  - Process facilitators

Performance Model

Pattern	Occurrences (#, %)		Average (h)	Min (h)	Max (h)	St. dev (h)
1 – 61 – 56	41422	49.5	255.14	8.52	1725.63	104.82
495 – 61 – 56	10250	12.3	22.82	6.24	96.69	12.64
1 – 44 – 61 – 56	4546	5.4	253.29	10.00	879.77	114.91
1 – 61 – 24 – 56	3495	4.2	346.05	49.81	948.34	123.58
1 – 61 – 24 – 45 – 56	3101	3.7	381.79	90.97	1132.73	125.67
1 – 61 – 71 – 56	1086	1.3	397.86	73.00	701.25	146.02
495 – 61 – 24 – 56	701	0.8	119.65	18.92	233.29	46.92
495 – 44 – 61 – 56	429	0.5	197.49	12.12	368.49	60.17
1 – 44 – 61 – 24 – 45 – 56	421	0.5	377.24	126.25	926.75	127.56
1 – 44 – 61 – 24 – 56	343	0.4	350.92	87.36	911.38	127.91
495 – 61 – 71 – 56	264	0.3	80.78	72.20	123.06	8.83
1 – 61 – 71 – 24 – 56	154	0.2	432.58	117.87	646.09	132.53
495 – 61 – 24 – 45 – 56	138	0.2	159.37	31.07	300.36	53.44
1 – 61 – 71 – 24 – 45 – 56	121	0.1	450.98	122.12	677.39	122.97
1 – 44 – 61 – 71 – 56	111	0.1	427.23	75.75	578.13	114.14
Total	66582	79.6				

- **Throughput times of traces**

Case conclusions

- From detailed log files we can extract information
  - Process model
  - Task transfer model
  - Performance model
- Models beyond the software: organizational context!
- Answering the question:
  - Does my company actually work the way I thought?

Conclusions

- **Reverse engineering = getting models from existing system**
- **Models are useful if they give additional knowledge about software system**
- **The choice of models depends on the task in hand (the knowledge we want to obtain)**
- **Visualisation is important BUT**
- **Numbers really matter**