LaQuSo
Laboratory for Quality Software

Quality Assessments on Source Code

at LaQuSo

LaQuSo
Laboratory for Quality Software

TU/e Technische Universiteit **Eindhoven** University of Technology

Where innovation starts

What is LaQuSo? What do we do?

LaQuSo

Laboratory for Quality Software

What is LaQuSo? What do we do?

LaQuSo

is for Quality Software

- a *lab* (TU/e, HG 5)
- + CS staff (TU/e)
- + CS staff (RU Nijmegen)

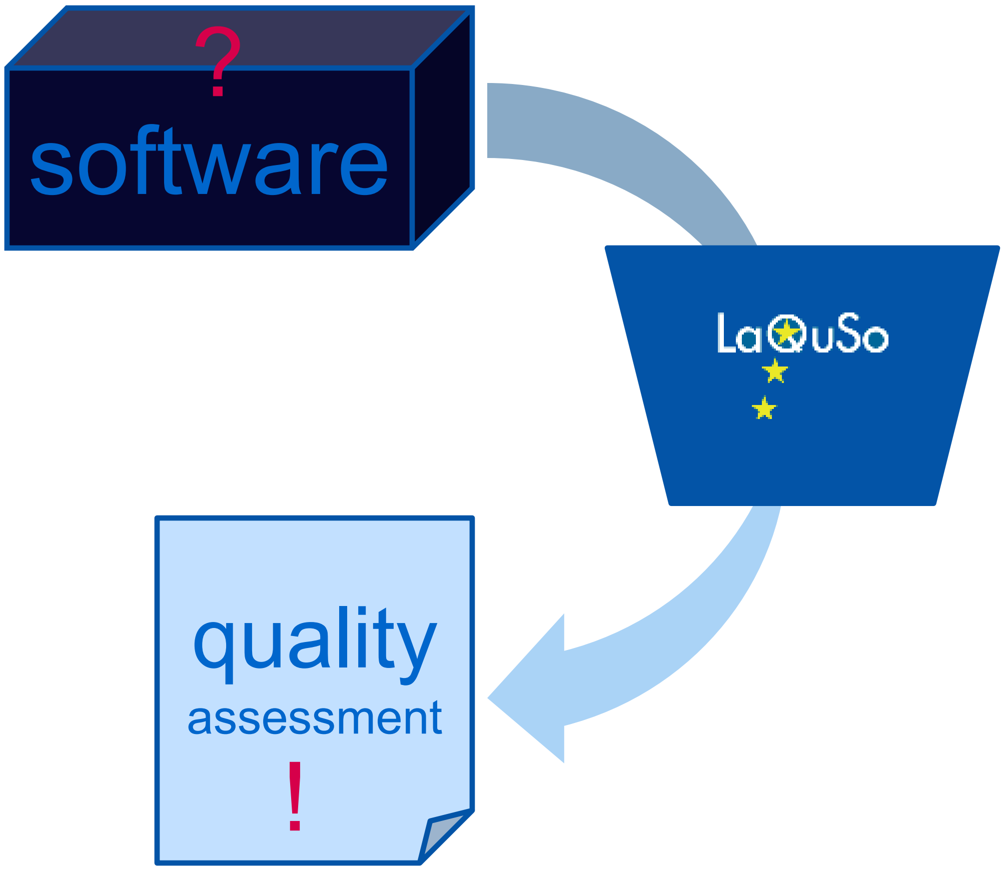
What is LaQuSo? What do we do?

LaQuSo

.... *assesses* Software Quality

(we also do other things)

Software quality assessment



Software quality assessment

lots of options for inputs, process, and outputs:

- **source code**
- **comments**

- running system (?)

- user/install docs (?)
- design docs (??)
- models (??)

tools + scripts:

- *analysis*
- *visualization*

LaQuSo

- *browsing ('expert review')*

- **code metrics**
- **dep. graph**
- **dupl. graph**

- code smells / sloppiness
- coding standards conformance

- process models
- dynamic properties

- arch. issues
- bugs (?)
- misc. issues

Software quality assessment

with *static*, not *dynamic* analysis

- **source code**
- **comments**

- ~~running system (?)~~

- user/install docs (?)
- design docs (??)
- models (??)

tools + scripts:
- *analysis*
- *visualization*

LaQuSo

- *browsing ('expert review')*

- **code metrics**
- **dep. graph**
- **dupl. graph**

- code smells / sloppiness
- coding standards conformance
- process models
- dynamic properties

- arch. issues
- bugs (?)
- misc. issues

Software quality assessment

with static, *structural*, not *behavioral* analysis

- **source code**
- **comments**

- user/install docs (?)
- design docs (??)
- models (??)

tools + scripts:

- *analysis*
- *visualization*

LaQuSo

- *browsing*
  ('expert review')

- **code metrics**
- **dep. graph**
- **dupl. graph**

- code smells / sloppiness
- coding standards conformance
- ~~process models~~
- ~~dynamic properties~~
- arch. issues
- bugs (?)
- misc. issues

Software quality assessment

with static, *structural*, not *behavioral* analysis

structure ("architecture"):

how is it put together?

assessed with

- **code quality metrics**
- **dependency graph analysis**
- **duplication graph analysis**

behavior:

what does it do?

assessed with

- **dataflow analysis**
- **assertion checking**
- **model checking**

Software quality assessment

on just source code (design documentation is rare)

?

- **source code**
- **comments**

- ~~user/install docs (?)~~
- ~~design docs (??)~~
- ~~models (??)~~

tools + scripts:

- *analysis*
- *visualization*

LaQuSo

- *browsing*
  ('expert review')

!

- **code metrics**
- **dep. graph**
- **dupl. graph**

- ...

Source code quality assessment

with static, structural analysis

- **source code**
- **comments**

tools + scripts:

- *analysis*
- *visualization*

LaQuSo

- *browsing ('expert review')*

?

- **code quality metrics**
- **dependency graph analysis**
- **duplication graph analysis**

!

Source code quality assessment

with static, structural analysis

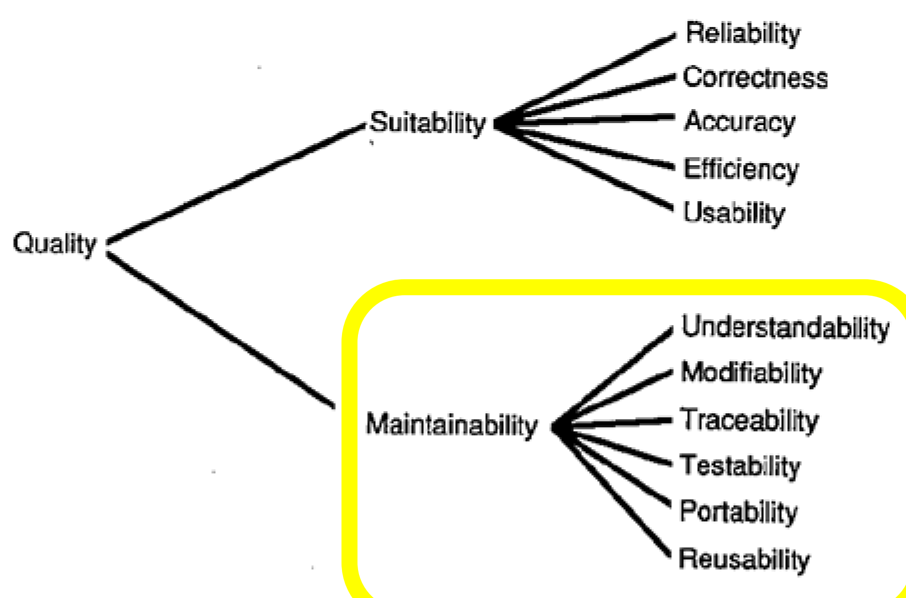


Figure 15.13 Some characteristics of software product quality.

- **End-User**:
  - behavior, performance, security, reliability
- **Customers**:
  - low cost, timely delivery
- **Product-Management**:
  - features, short time to market, low cost, parity with products
- **Development**:
  - low cost, employability
- **Maintenance**:
  - modifiability

Stakeholders

from: *Object-Oriented Software Engineering*
A Use Case Driven Approach
Ivar Jacobson et al., Addison-Wesley, 1992

from: *last hour*
R. Bril

What is maintainability?

- **What is maintenance?**
  - To fix an error / bug in the system
  - To add a new feature to the system
  - To adapt the system to a new environment

- **Why check maintainability?**
  - Better maintainable systems take less time and less money to adapt and fix

What code is hard to maintain?

- Poorly understandable
  - not documented
  - cluttered or inconsistently used/developed code
  - too big
- Poorly modifiable
  - code is duplicated
  - code is intertwined
  - code is non-extendable
  - code is non-portable
- Poorly testable / analysable
  - code is too complex

Source code structure

structure?

essential

vs.

accidental

complexity

Source code quality assessment

with static, structural analysis

structure?

essential

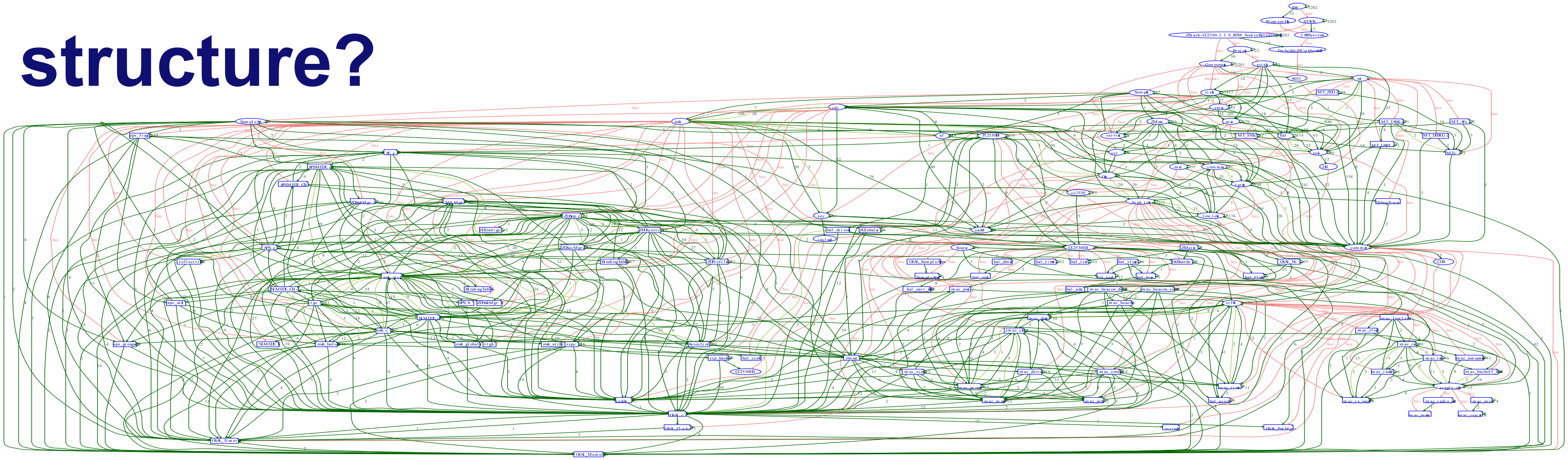
vs.

accidental

complexity

Source code structure

structure?



Source code structure

structure?

software entropy

Assume that a system initially has a certain software entropy. Experience shows that it is reasonable to assume that the increase in software entropy is proportional to the entropy of the software when the modification started. This means that it is easier to change an ordered system than a disordered one, something that all experience shows. This would mathematically be expressed as

$$\Delta E \sim E$$

or, with differential calculus

$$\frac{dE}{dt} = kE$$

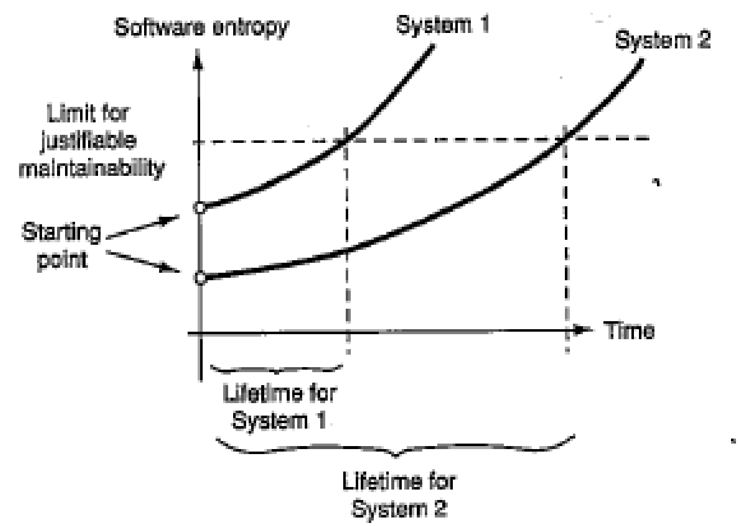


Figure 4.1 A system's entropy and how it increases at different speeds depending on the starting entropy.

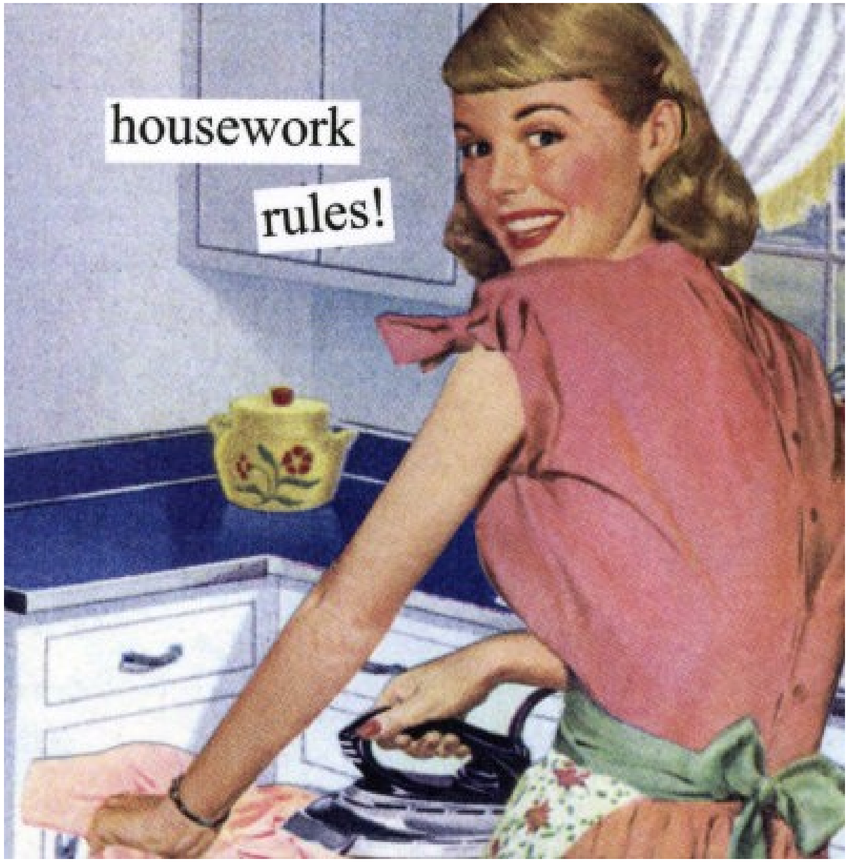
from: *Object-Oriented Software Engineering, A Use Case Driven Approach*
Ivar Jacobson et al., Addison-Wesley, 1992

Source code structure

structure?

tidy your room, dear

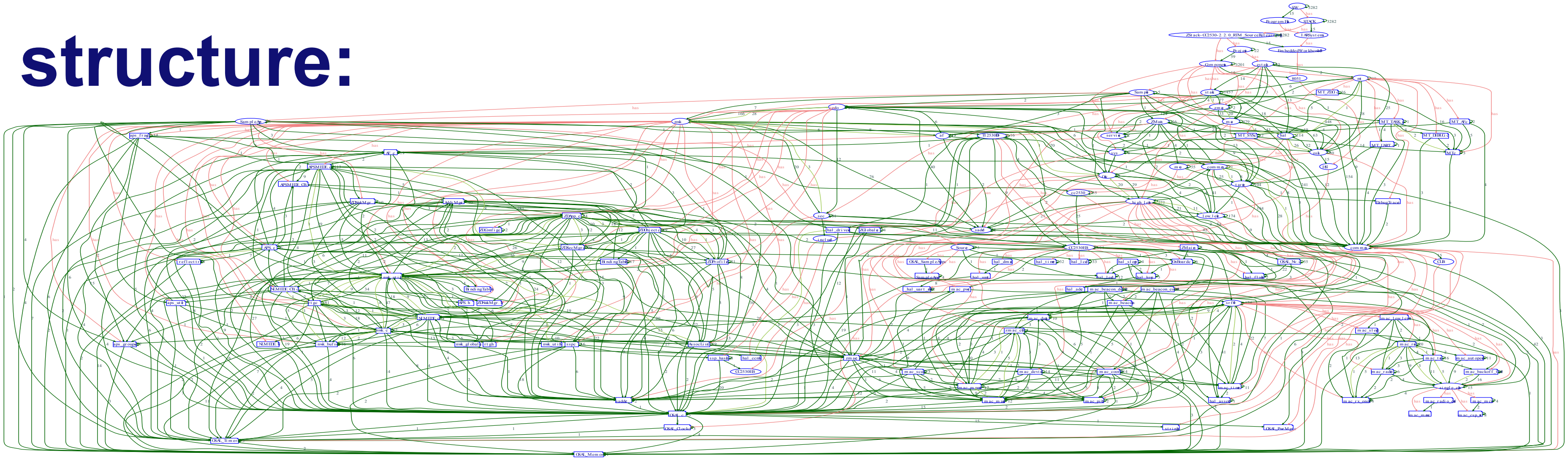
from: *(personal communication),*
mother, yesterday



Source code structure

LaQuSo's job:

structure:



- map out the structure
- map out + measure the mess (if any)

Source code maintainability
the role of source code quality metrics

metrics related to software entropy:

- number of lines per file / function
- number of *code* lines per file / function
- McCabe complexity per function
- Halstead development effort
- percentage of duplicated code per file / function
- fan-in / fan-out based metrics per class / package

other metrics:

- percentage of comments per file

LaQuSo's results (2004-2009) in source code assessment

- **Many successful assessments for companies**
  (from one-man companies to multinationals)
- **Assessment tools built / integrated**
- **Scientific studies on assessments**

Summary

- LaQuSo assesses software quality
- usually maintainability
- usually based on source code only
- usually focusing on architecture, not behavior

- the architectural structure is visualized
- its tidiness is measured
- untidiness is manually inspected

LaQuSo's results

A case study

:

“How maintainable is our system?”

Case study for a financial organization

Question: ”*How maintainable is our system?*”

(Shall we continue maintenance for another 5 years?)

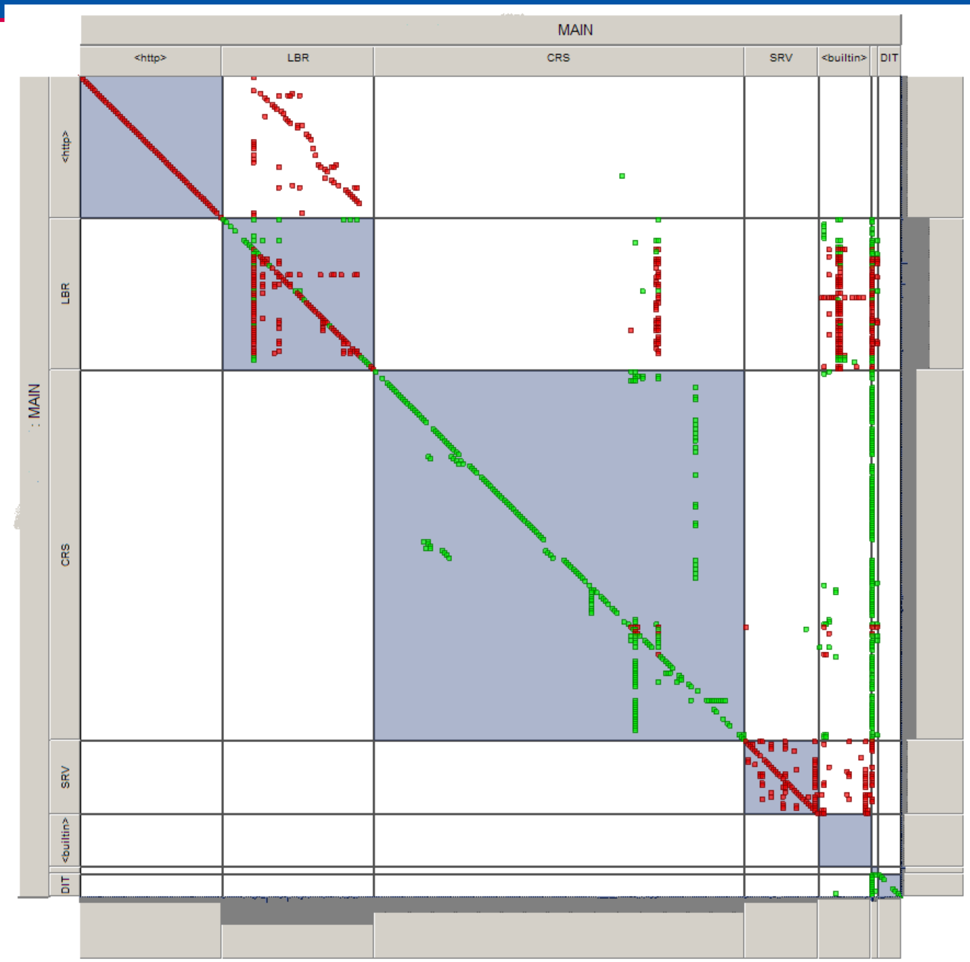
Facts:

- **System of approximately 15 years old**
- **Web application**
  - client side : HTML, JavaScript, some Java
  - server side: PL/SQL, some Java
- **Other languages involved**
  - C, COBOL, Oracle Forms
  - Links through common use of the database
- **Very limited documentation**

“How maintainable is our system?”

Case study for a financial organization

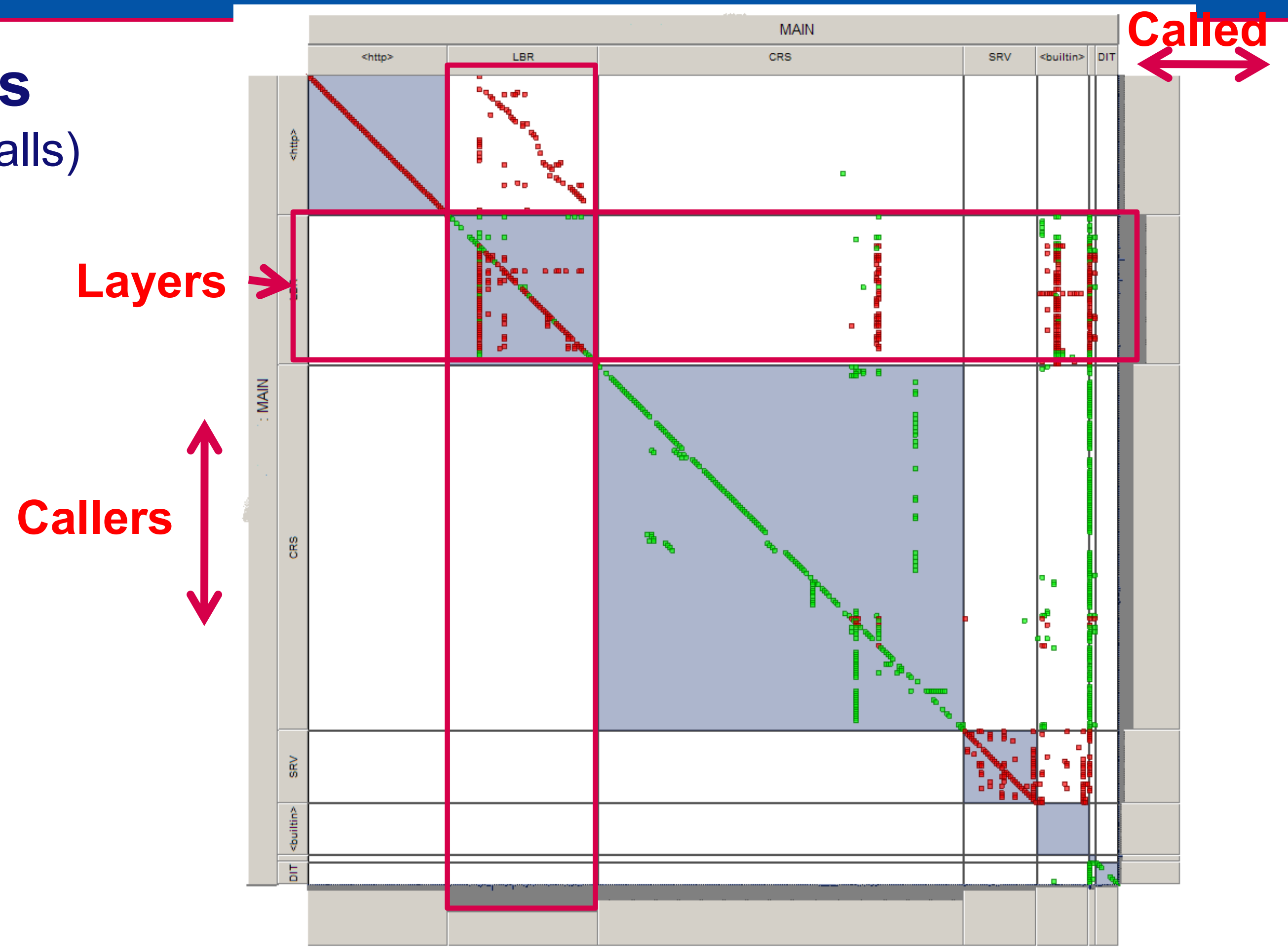
- **Dependencies**



"How maintainable is our system?"

Case study for a financial organization

- **Dependencies**
  (method/function calls)



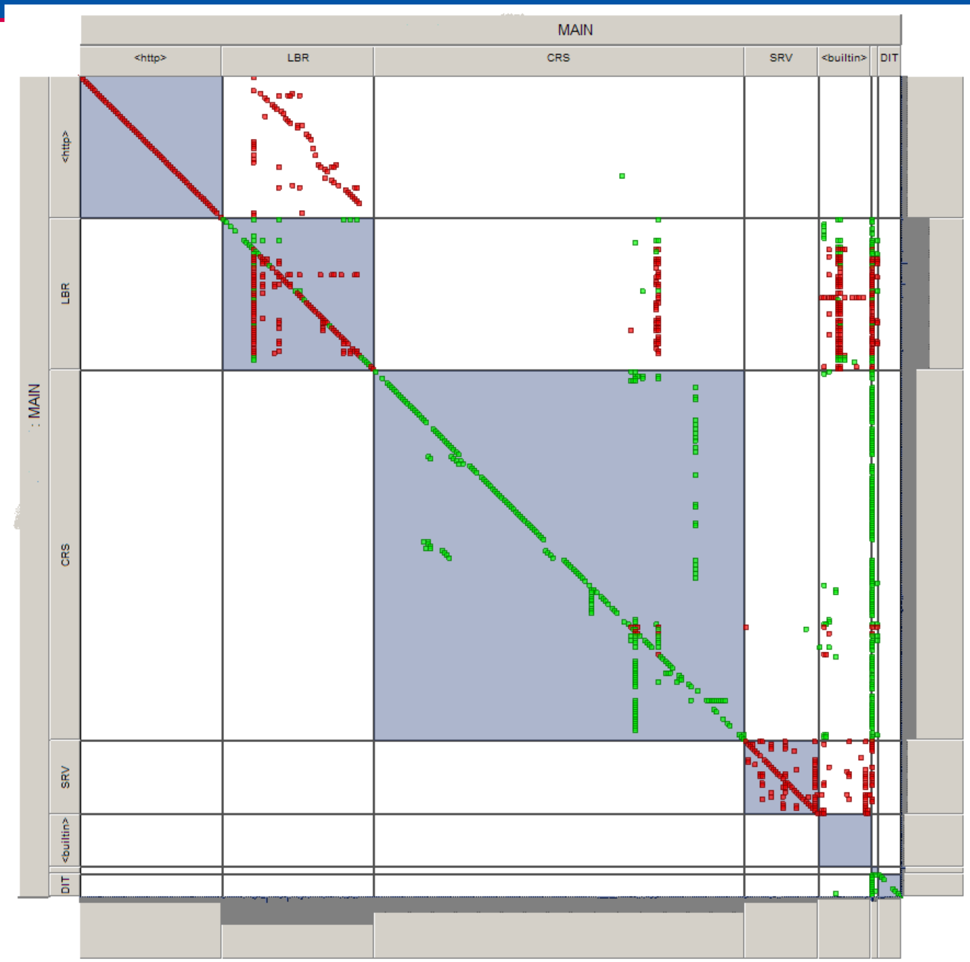
"How maintainable is our system?"

Case study for a financial organization

- **Dependencies**

Red: Calls from and to modules inside the system of interest

Green: Calls from and to modules outside the system of interest



“How maintainable is our system?”

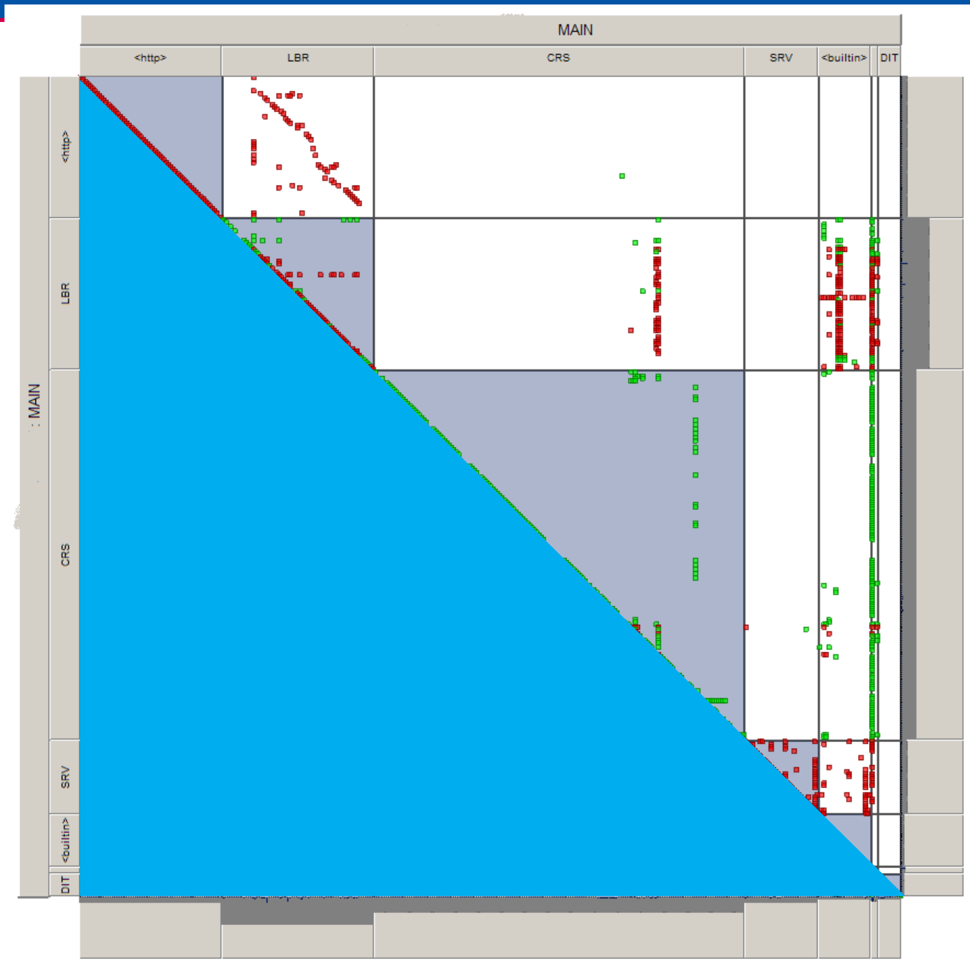
Case study for a financial organization

- **Dependencies**

Red: Calls from and to modules inside the system of interest

Green: Calls from and to modules outside the system of interest

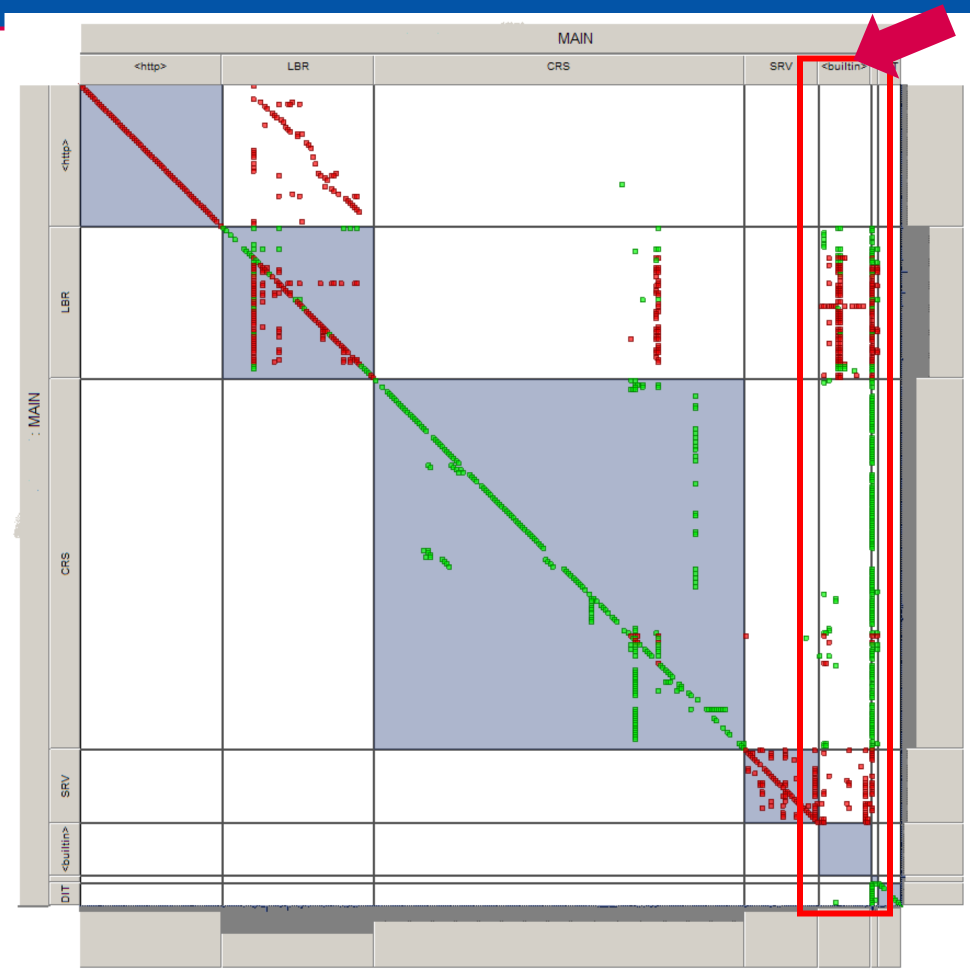
Calls in highlighted area break layering rules



"How maintainable is our system?"

Case study for a financial organization

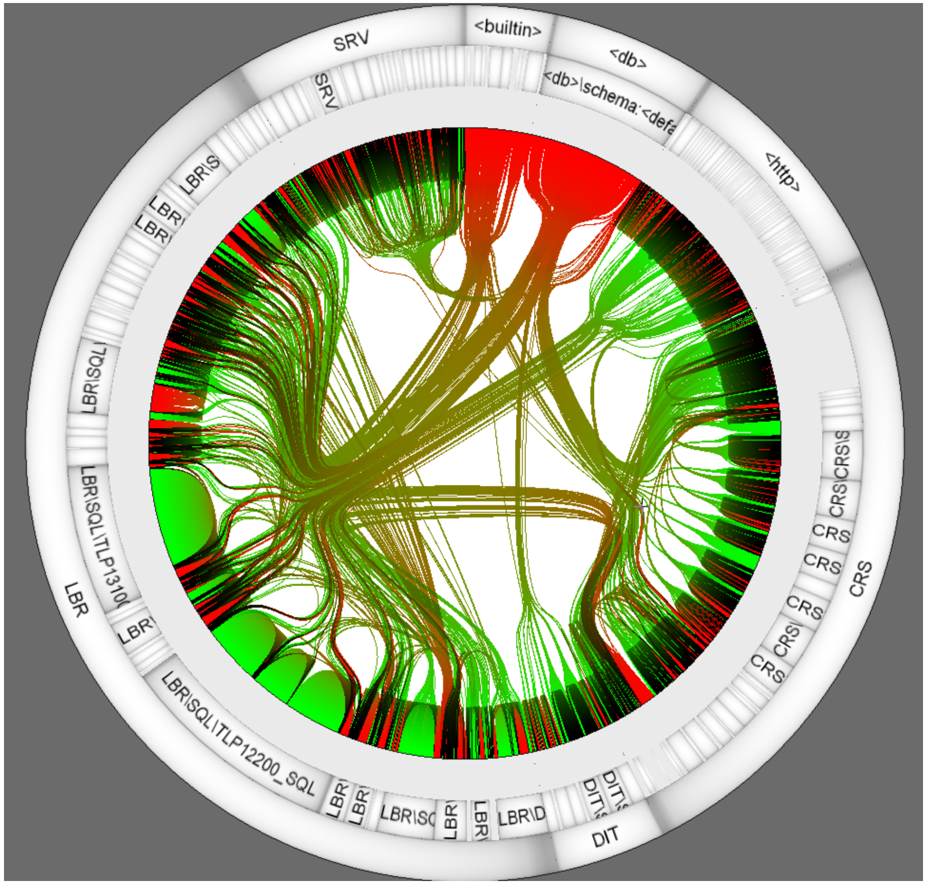
- Red arrow = data layer
- Data layer only *receives*
- Almost layered architecture
- Good design, however…
- The data layer is accessed from several other layers



“How maintainable is our system?”

Case study for a financial organization

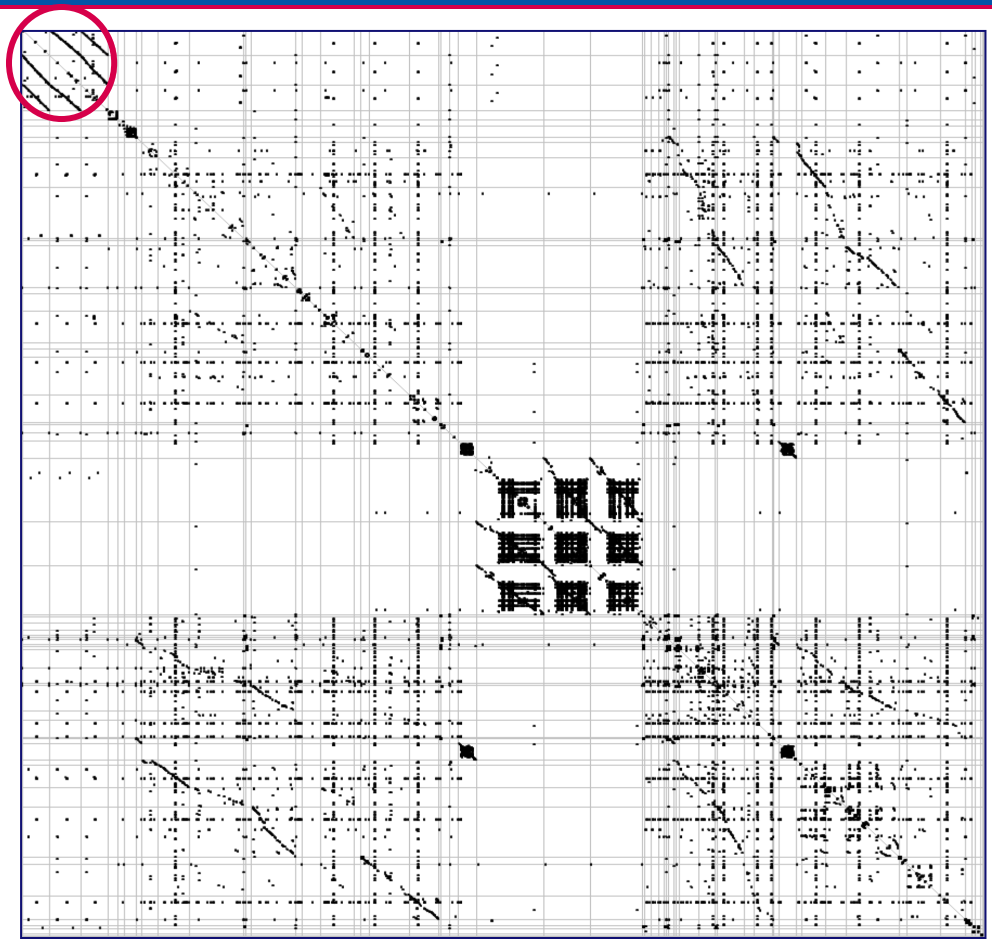
- **Dependencies**
  - **Visualizing the calls between modules**
  - **By expanding and collapsing, we can identify individual faulty dependencies**
  - **Huge green ‘bubbles’ reflect many internal calls**



"How maintainable is our system?"

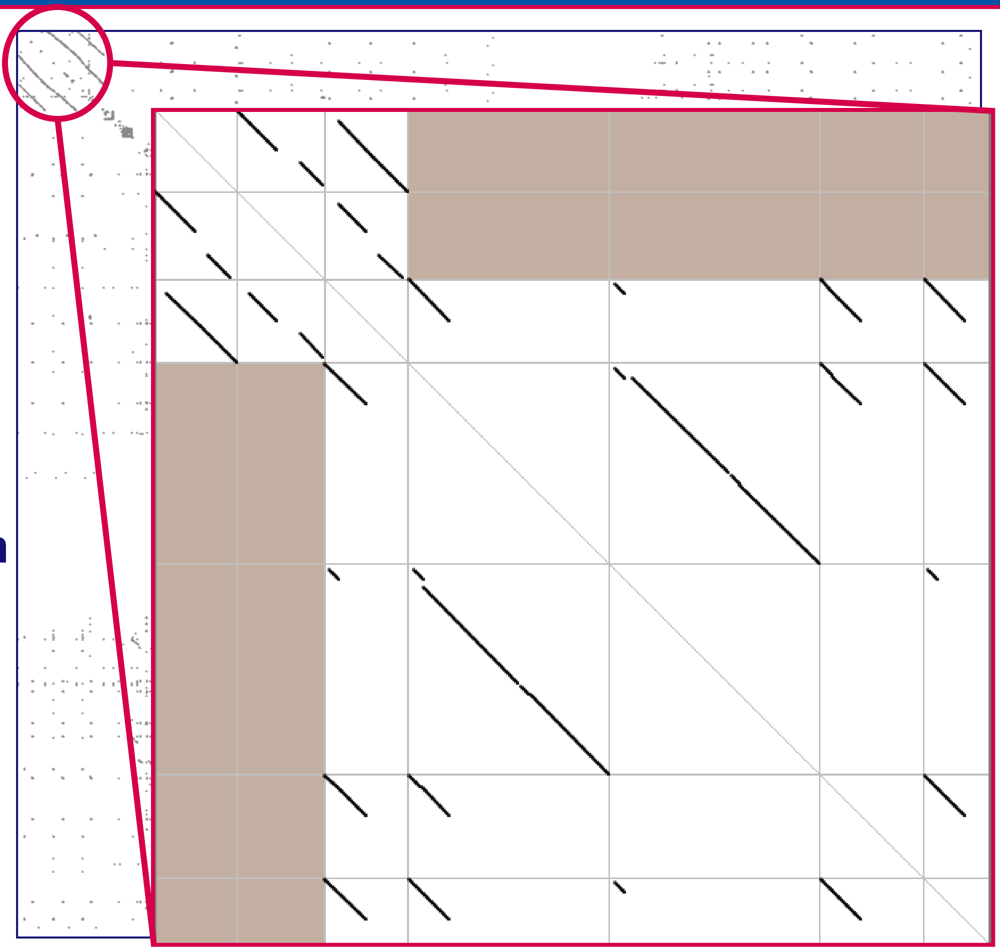
Case study for a financial organization

- **Code duplication**
  - **Many occurrences of code duplication found**



"How maintainable is our system?"
Case study for a financial organization

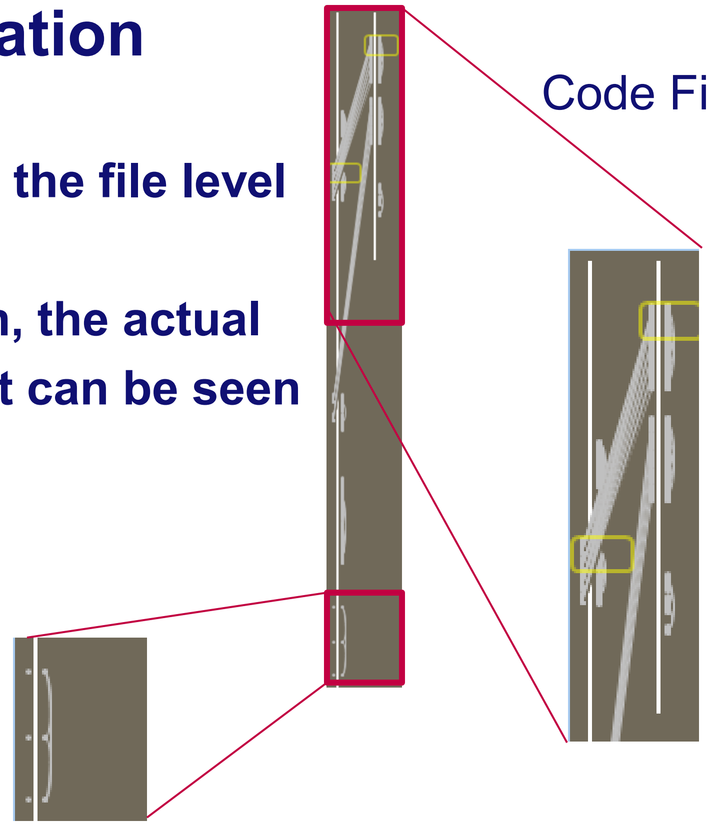
- **Code duplication**
  - **Large parts of files are present in other files**
  - **By zooming in, the actual code fragment can be seen**



"How maintainable is our system?"

Case study for a financial organization

- **Code duplication**
  - **Zooming in to the file level**
  - **By zooming in, the actual code fragment can be seen**



- **Code commenting**
  - **extensive**
  - **thorough (explains design and implementation decisions)**

“How maintainable is our system?”

Case study for a financial organization

Findings:

The system is well structured
(layered architecture)

Code duplication pollutes the system
(refactor on further development)

A list of strong and weak points with recommendations

We can estimate annual maintenance effort
(Halstead effort, function points)

LaQuSo's results

Tool demo

:

SQuAVisiT tool demo.swf

Summary

- **Maintenance costs time and money**
  to fix, add and adapt features in systems

- **How much depends on the quality of the system**

- **Code quality assessment (“code mining”) can be used:**
  - as an overall health check of the system
  - as aid for solving specific problems
  - for getting insight in the architecture and system internals

- **LaQuSo has tooling for multiple languages and visualizations**

What does LaQuSo do?

- **Code quality assessment ("code mining")**
  - **Assessing overall quality, performance, maintainability and reliability of code bases**
- Process mining
  - Reverse engineering processes and fact extraction from running systems
- Model analysis
  - Discovering critical behavioral errors in design and code
- Independent assessments
  - Independent assessment and certification of a software artifact (requirements, design, code, tests, documentation)

LaQuSo's results

More case studies

:

“Is architectural purity preserved?”

Case study for a embedded systems manufacturer

Question: ” *With extensive changes to the system, is architectural purity still preserved?”*

(Developers assume that the architecture is layered)

Facts:

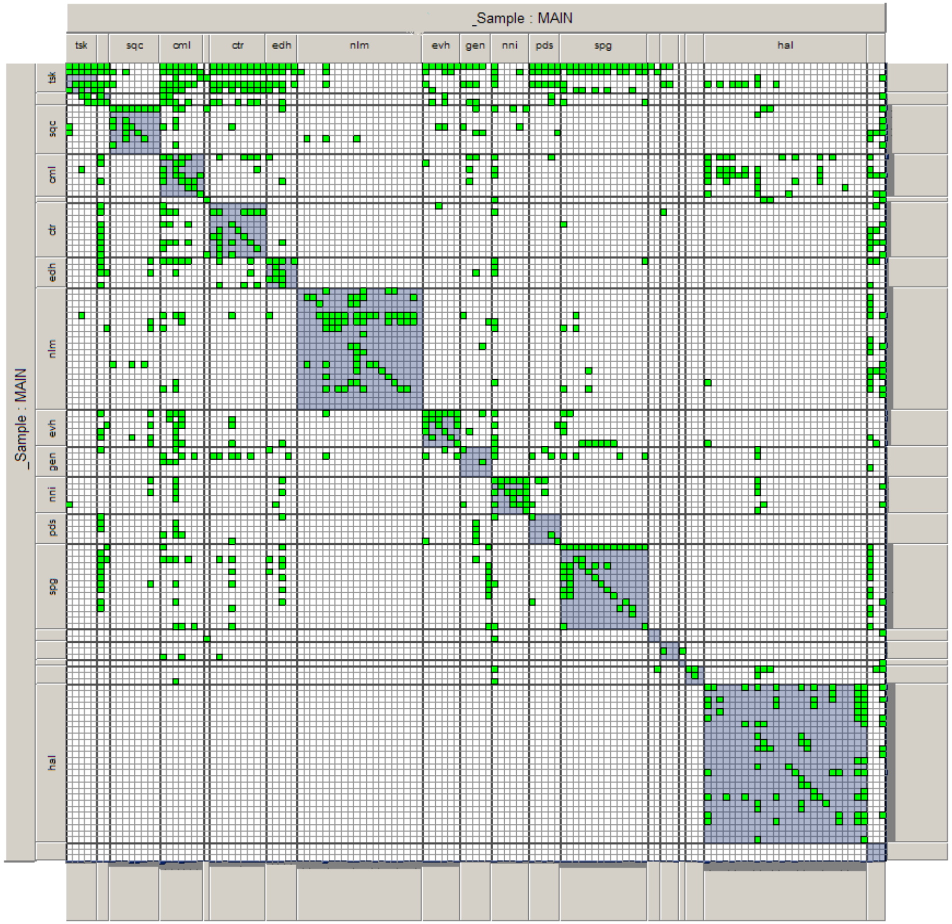
- Component system with compile-time binding via make files
- C with embedded Assembler
- 6 years old
- Medium size of 150 KLoC
- No access to documentation

“Is architectural purity preserved?”

Case study for a embedded systems manufacturer

- **Dependencies**

- **Visualization of caller and called dependencies**



“Is architectural purity preserved?”

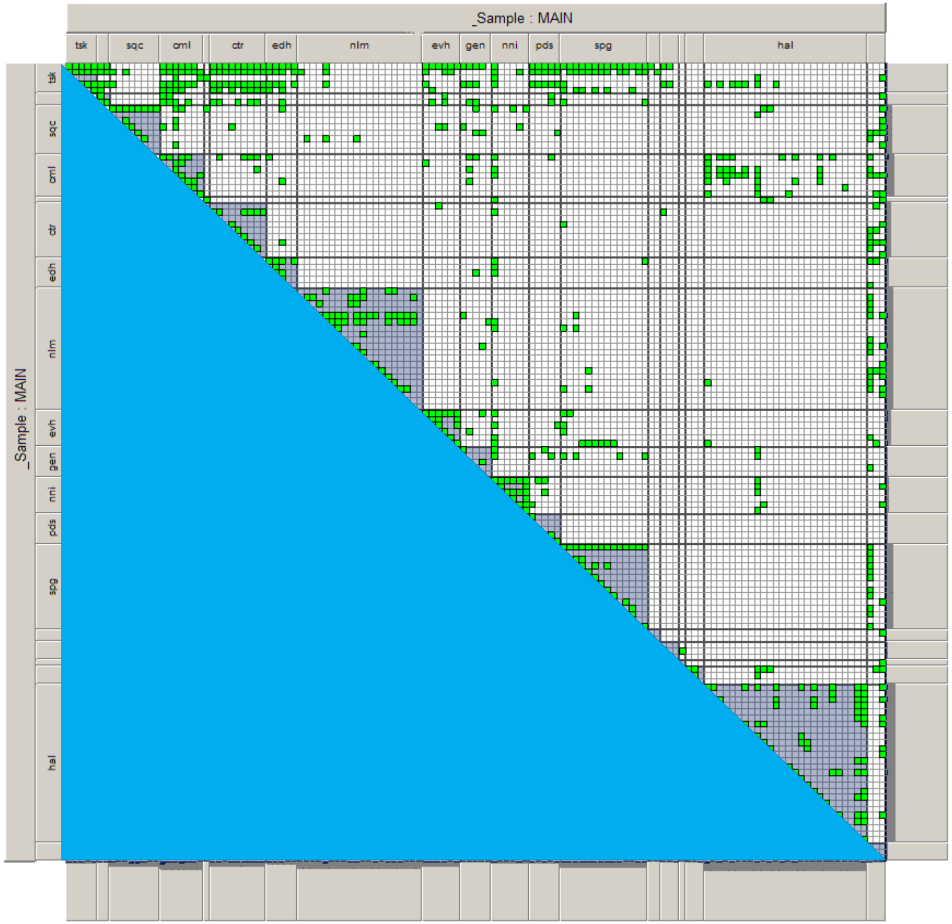
Case study for a embedded systems manufacturer

- **Dependencies**

  - **Visualization of caller and called dependencies**

  - **Visualization of the architectural dependencies shows unlayered architecture**

Calls in highlighted area break layering rules



“Is architectural purity preserved?”

Case study for a embedded systems manufacturer

<u>Findings</u>:

The system is poorly layered

Unexpected cross dependencies exist between components

Extensive changes to the system will put even more stress on the architecture

“Why is it so slow?”

Case study for a pension fund

Question: *“The calculation for creating the annual survey takes very long. Why is this?”*

“What is the quality of the architecture?”

(migration at hand due to discontinuation of support)

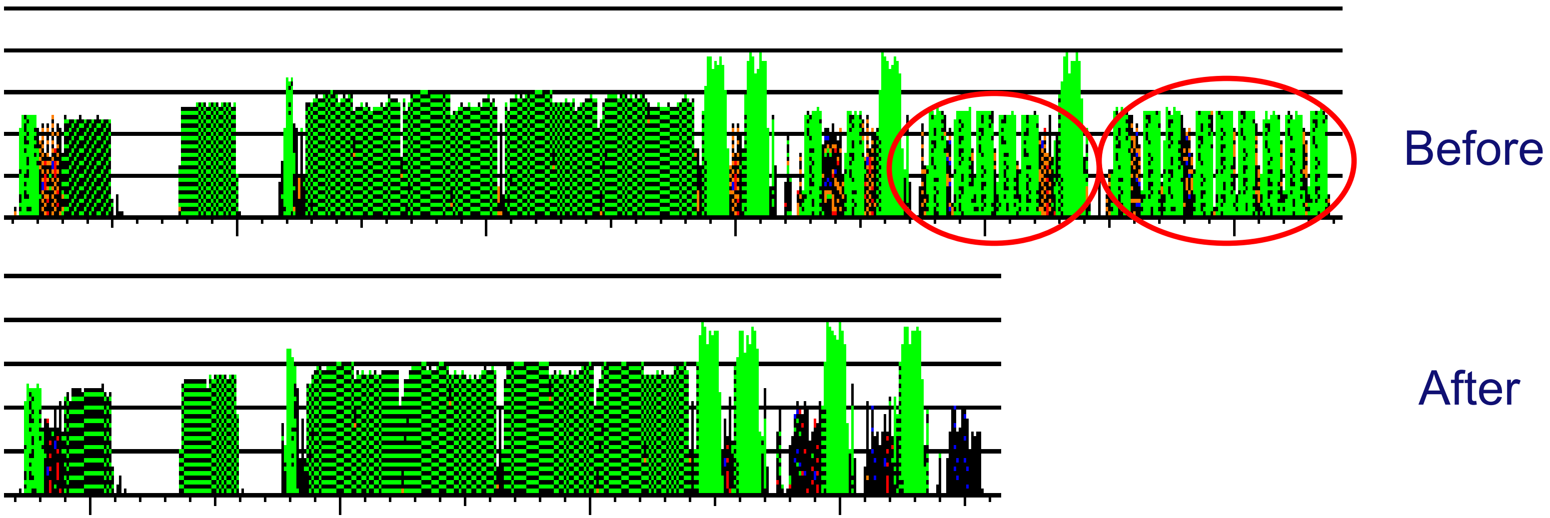
Facts:

- Homogeneous system in COBOL
- 17 years old
- Large system of 1.7 MLoC
- Communication with an Oracle 9*i* database

"Why is it so slow?"

Case study for a pension fund

- **Unnecessary querying to the database discovered**
  - **By visualizing queries, patterns emerge:**

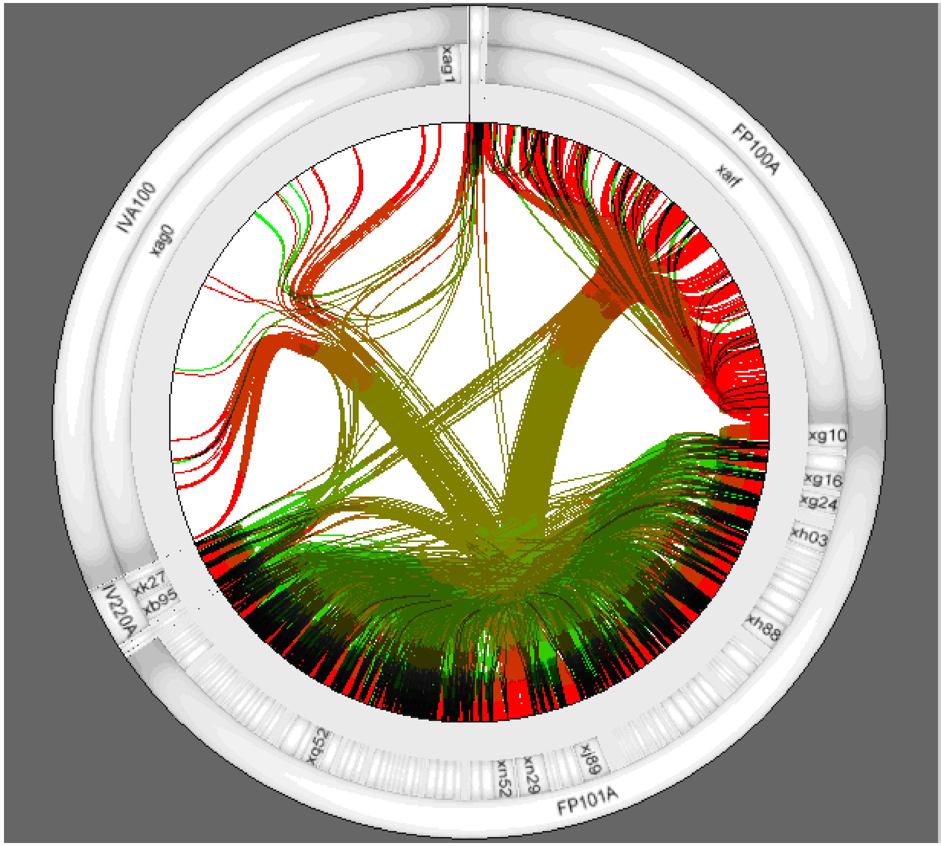


Increase in speed: 40%

“Can we port it? Is the architecture tidy?”

Case study for a pension fund

- **Dependencies**
  - **A lot of open spaces**
  - **1216 modules not called by other modules**
  - **This may be dead code**
  - **651 modules indeed dead (confirmed)**

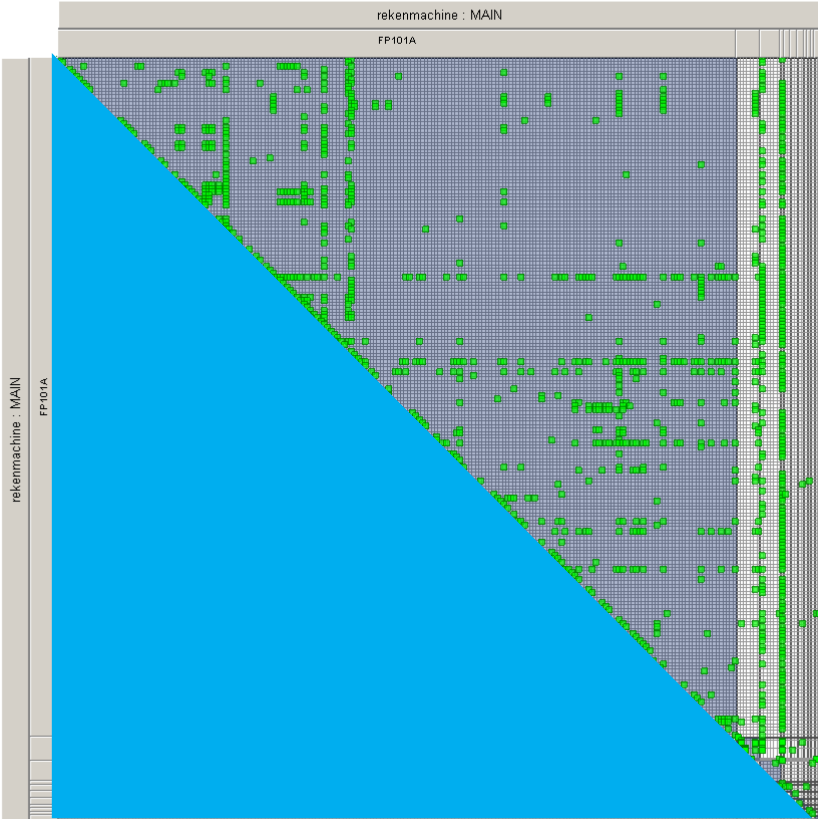


"Can we port it? Is the architecture tidy?"

Case study for a pension fund

- **Dependencies**

  - **Many violations in layering**



“Can we port it? Is the architecture tidy?”

Case study for a pension fund

- **Calculating quality metrics on the source code**

Fan Out *(# modules called)*

Layer	Unit	Module	LOCs	omment	Blanks	Source	IFs	LOOPs	McCabe	Fan_in	Fan_out	CLN	NBR	RSA	RSI	CVR	id
'CobolPr	'XOFC'	'XOFC27	17480	2333	310	14837	256	63	320	0	21	768	8	0.27200	0.60300	0.71600	0
'CobolPr	'XOFC'	'XOFC27	16331	1627	463	14241	194	105	300	1	53	887	8	0.53000	0.55900	0.71700	1
'CobolPr	'XOFC'	'XOFC27	8722	707	313	7702	105	25	131	1	20	715	8	0.82900	0.51500	0.86100	3
'CobolPr	'XOFC'	'XOFC27	11391	911	598	9882	87	26	114	1	31	771	8	0.69600	0.49500	0.73000	2
'CobolPr	'XUFC'	'XUFC06	1688	249	104	1335	47	2	50	1	0	66	8	0.21700	0.19400	0.27200	8
'CobolPr	'XIFC'	'XIFC050	1100	147	65	888	33	5	39	1	5	25	8	0.15900	0.51700	0.59900	6
'CobolPr	'XJFC'	'XJFC630	3000	185	199	2616	11	12	24	1	7	176	8	0.27600	0.51400	0.66200	4
'Rekenre	'XARF'	'XARF03	384	49	31	304	10	0	11	1	0	7	4	0.09800	0.34300	0.44000	14
'Datalaye	'XU06'	'XU0610	671	106	56	509	8	0	9	1	0	4	2	0.02300	0.63000	0.64500	11
'CobolPr	'XNFC'	'XNFC86	1112	133	143	836	6	0	7	1	6	115	8	0.54300	0.40900	0.56400	5
'CobolPr	'XNFC'	'XNFC86	630	109	52	469	2	1	4	1	8	48	8	0.37900	0.16700	0.37900	7
'Datalaye	'XU06'	'XU0610	180	36	34	110	2	0	3	1	0	2	2	0.13900	0.00000	0.13900	9
'Rekenre	'XARF'	'XARF02	130	10	28	92	2	0	3	1	0	3	4	0.48900	0.00000	0.48900	13
'Datalaye	'XU06'	'XU0610	177	41	33	103	2	0	3	1	0	2	2	0.13900	0.00000	0.13900	10
'Rekenre	'XARF'	'XARF03	119	10	28	81	0	0	1	1	0	7	4	0.94800	0.00000	0.94800	15
'Rekenre	'XARF'	'XARF03	117	10	28	79	0	0	1	1	0	3	4	0.59800	0.00000	0.59800	16
'Rekenre	'XARF'	'XARF03	119	10	28	81	0	0	1	1	0	7	4	0.94300	0.00000	0.94300	12

McCabe complexity

(#If’s + #Loops +1)

"Can we port it? Is the architecture tidy?"

Case study for a pension fund

- **Calculating quality metrics on the source code**

Fan Out *(# modules called)*

<u>Guideline:</u> McCabe ≤ 30

Some are over 100 going up to 320!

This rules out white-box testing

[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	IFs	LOOPs	McCabe	Fan_in	Fan_out	CLN	NBR	RSA	RSI	CVR	id
[illegible]	[illegible]	[illegible]	[illegible]	2333	310	14837	256	63	320	0	21	768	8	0.27200	0.60300	0.71600	0
'CobolPr	'XOFC'	'XOFC27	16331	1627	463	14241	194	105	300	1	53	887	8	0.53000	0.55900	0.71700	1
'CobolPr	'XOFC'	'XOFC27	8722	707	313	7702	105	25	131	1	20	715	8	0.82900	0.51500	0.86100	3
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	31	771	8	0.69600	0.49500	0.73000	2
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	0	66	8	0.21700	0.19400	0.27200	8
'CobolPr	'XIFC'	'XIFC050	1100	147	65	888	33	5	39	1	5	25	8	0.15900	0.51700	0.59900	6
'CobolPr	'XJFC'	'XJFC630	3000	185	199	2616	11	12	24	1	7	176	8	0.27600	0.51400	0.66200	4
'Rekenre	'XARF'	'XARF03	384	49	31	304	10	0	11	1	0	7	4	0.09800	0.34300	0.44000	14
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	1	0	4	2	0.02300	0.63000	0.64500	11
'CobolPr	'XNFC'	'XNFC86	1112	133	143	836	6	0	7	1	6	115	8	0.54300	0.40900	0.56400	5
'CobolPr	'XNFC'	'XNFC86	630	109	52	469	2	1	4	1	8	48	8	0.37900	0.16700	0.37900	7
'Datalaye	'XU06'	'XU0610	180	36	34	110	2	0	3	1	0	2	2	0.13900	0.00000	0.13900	9
'Rekenre	'XARF'	'XARF02	130	10	28	92	2	0	3	1	0	3	4	0.48900	0.00000	0.48900	13
'Datalaye	'XU06'	'XU0610	177	41	33	103	2	0	3	1	0	2	2	0.13900	0.00000	0.13900	10
'Rekenre	'XARF'	'XARF03	119	10	28	81	0	0	1	1	0	7	4	0.94800	0.00000	0.94800	15
'Rekenre	'XARF'	'XARF03	117	10	28	79	0	0	1	1	0	3	4	0.59800	0.00000	0.59800	16
'Rekenre	'XARF'	'XARF03	119	10	28	81	0	0	1	1	0	7	4	0.94300	0.00000	0.94300	12

Metrics can find maintenance landmines

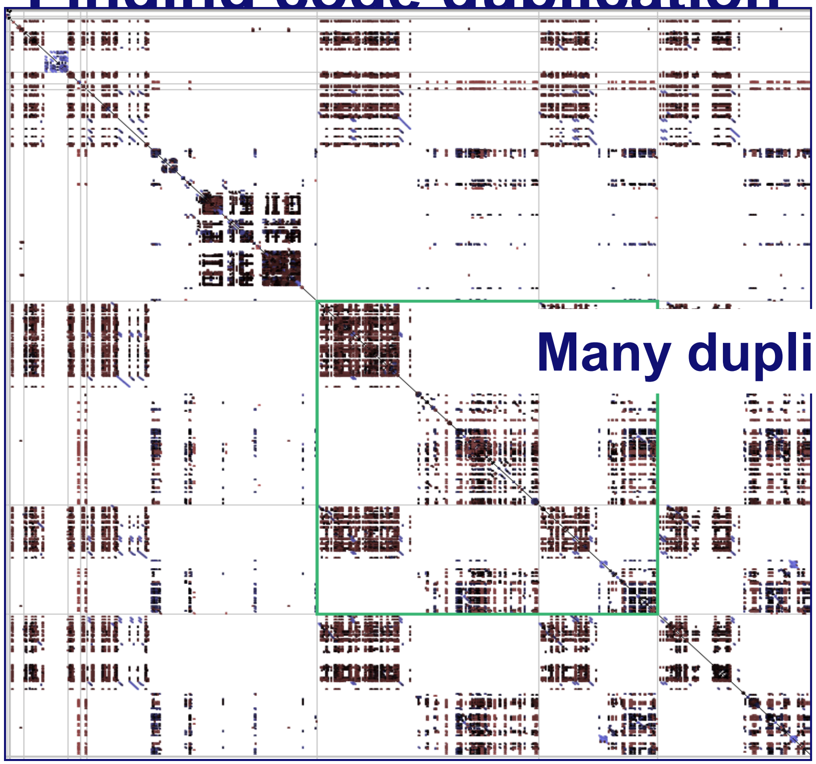
McCabe complexity

(#If's + #Loops +1)

“Can we port it? Is the architecture tidy?”

Case study for a pension fund

- **Finding code duplication**



Many duplications found!

“Can we port it? Is the architecture tidy?”

Case study for a pension fund

Findings

Reduction of unnecessary queries resulted in a 40% increase in speed in calculating annual surveys

651 confirmed dead modules and some modules are too complex based on metrics

No strict layering present in the architecture

With these findings, recommendations can be made for migration

“What’s the design of our system?”

Case study for a printer manufacturer

Question: *”What’s the design of our system?”*

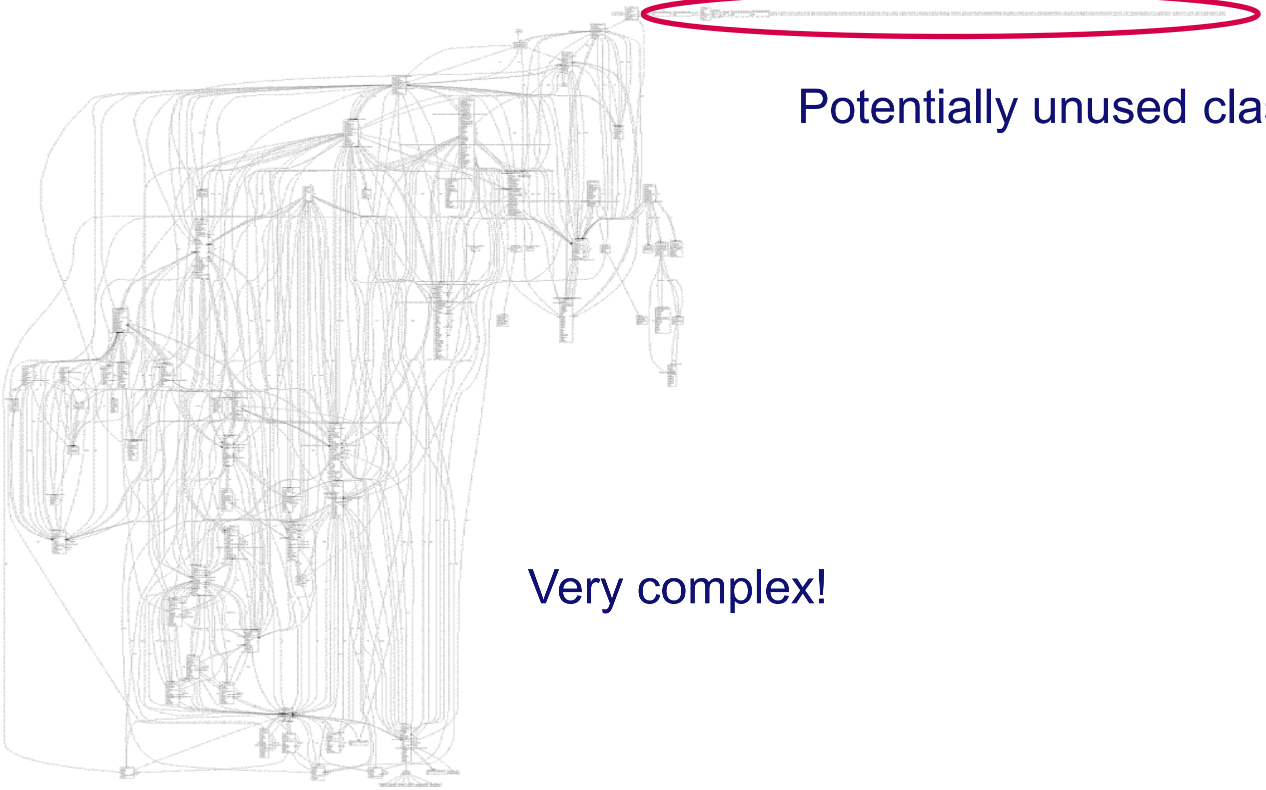
Facts:

- No documentation of system available
- C++ code
- 60,000 Lines of code
- We would like UML class and sequence diagrams

“What’s the design of our system?”

Case study for a printer manufacturer

Automatic model extraction shows:



Potentially unused classes!

Very complex!

“What’s the design of our system?”

Case study for a printer manufacturer

Class Diagrams

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

“What’s the design of our system?”

Case study for a printer manufacturer

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

“What’s the design of our system?”

Case study for a printer manufacturer

Findings

Automatic UML model extraction can help in understanding the system

Metrics on the acquired models can point out maintainability landmines

What else does LaQuSo do?

- Automatic model extraction
- Static analysis of source code
- Other types of visualizations of systems
- Estimate understandability, maintenance effort, etc.
- …

Model extraction

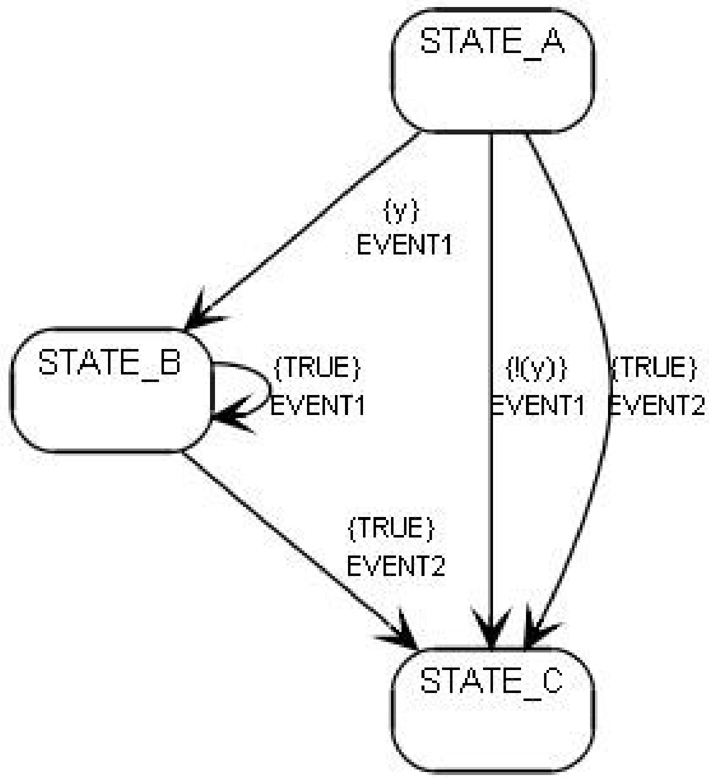
Extracting UML state diagrams from embedded C/C++ code

- State diagrams can be extracted from C/C++ code
- Transitions between states are guarded
- Based on alternative paths (if-then-else and switch) in the code as often seen in embedded software
- Extraction of models is fully automatic

Model extraction

Extracting UML state diagrams from embedded C/C++ code

```
static void OBJ_control(Obj *obj, ObjEvent event)
{bool y = true;
…
switch (obj->state) {
  case STATE_A:
    switch (event) {
      case EVENT1:
        if (y){obj->state = STATE_B; }
        else{ obj->state = STATE_C;}
      break;
      case EVENT2:
        …
        obj->state = STATE_C;
        …
        break;}
    break;
  case STATE_B:
    switch (event) {
      case EVENT1:
        …
        obj->state = STATE_A;
        …
        break;
      case EVENT3:
        …
        obj->state = STATE_C;
        …
        break;}
    break;
  default;}}
```



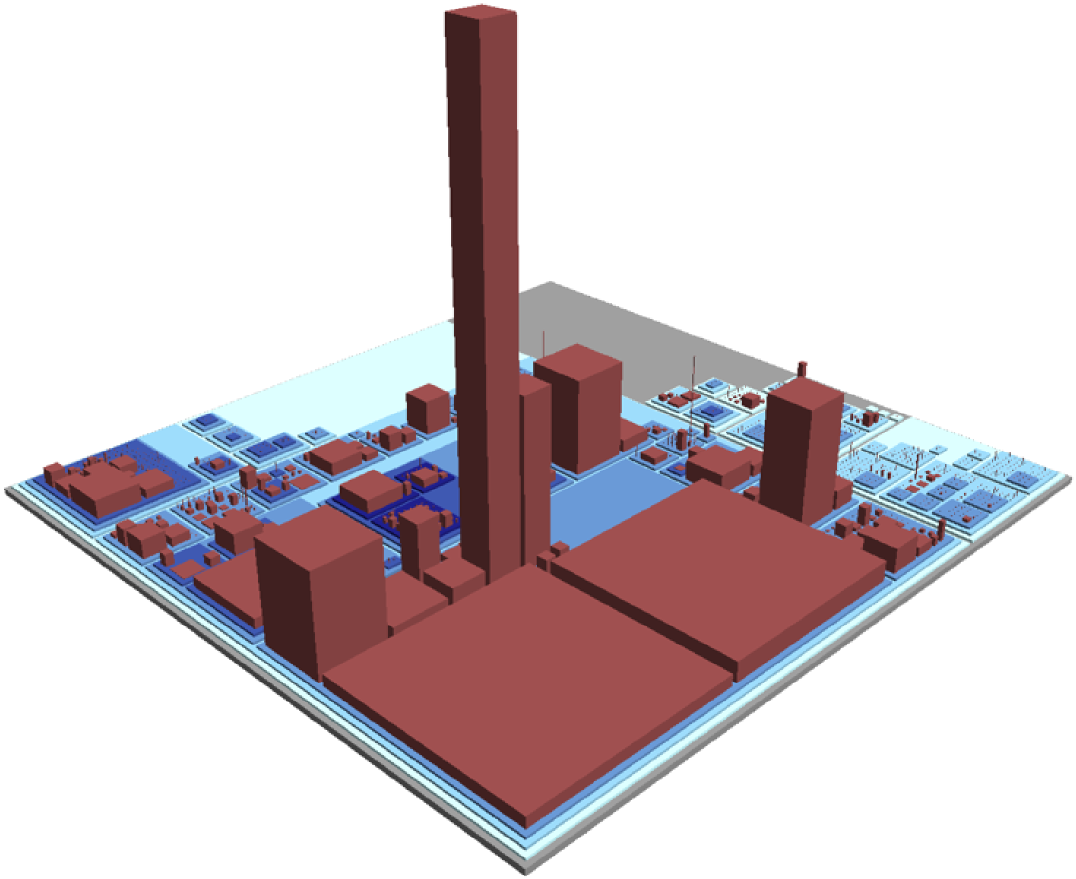
Static analysis of source code

- Automatically check source code for
  - Uninitialized variables
  - Null pointer dereferencing
  - Out of bounds referencing of arrays
  - User defined properties, e.g.
    - Lock – Unlock
    - B may only occur after A

Visualizing object–oriented systems

Visualizing structure

- **Base size:**
  Number of attributes
- **Height:**
  Number of methods



“Our load balance system crashes”

Case study for a printing service organization

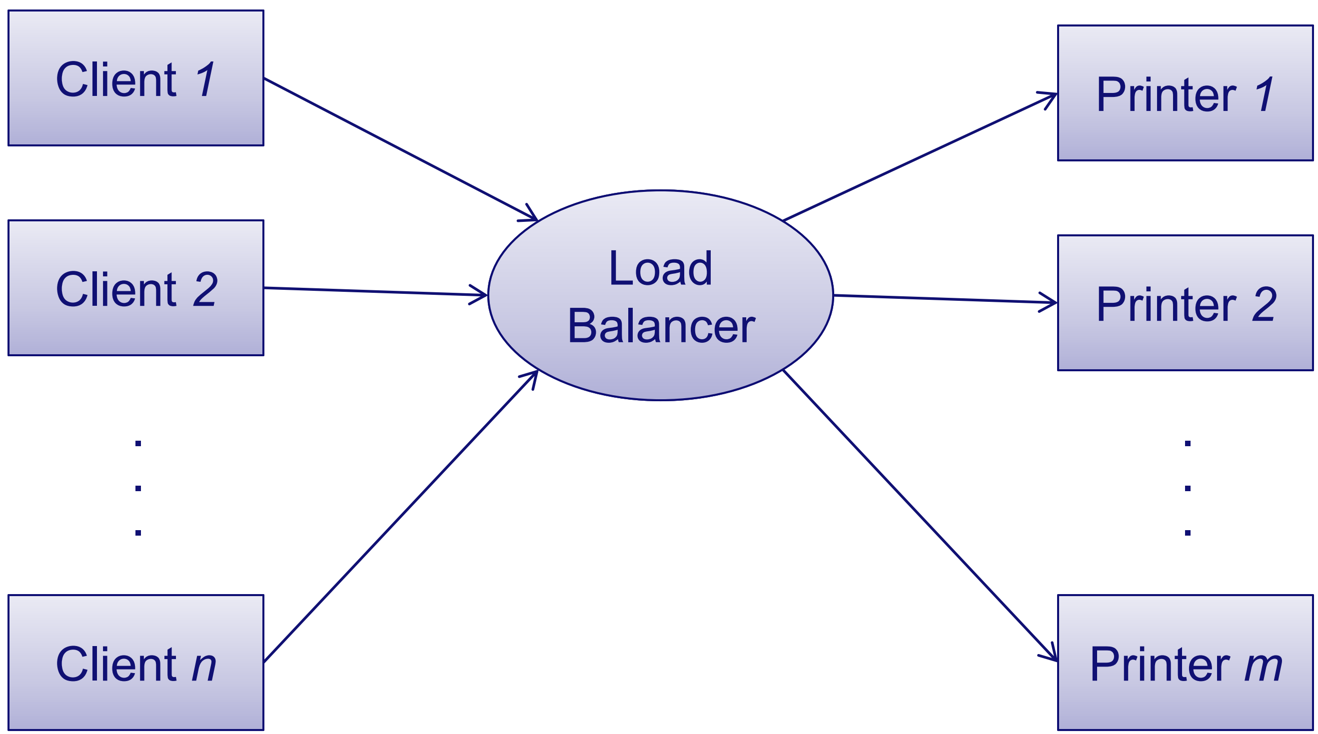
Problem: ***“Load balance system crashes spuriously”***

Facts:

- **Distribution of print jobs over document printers**
- **7,500 LoC in C language**

“Our load balance system crashes”

Case study for a printing service organization



“Our load balance system crashes”

Case study for a printing service organization

- The source code was manually translated into a mathematical model describing the behavior of the system
- This model can be checked fully automatically for unwanted behavioral properties
  - Free from deadlocks
  - Limits on locking
  - Limits on the number of requests
  - …

“Our load balance system crashes”

Case study for a printing service organization

- **The mathematical model is based on process algebra.**
- **Resulting model is complex:**

#Clients	#Servers	Time	#Levels	#States	#Transi-tions
1	1	7m38s	241	657k	1.38M
1	2	3h01m	267	18M	38.5M
2	3	9h55m	444	54M	141M
1	3	13h*	481	213M	465.5M
2	2	>113h*	>215	>511M	>1121M

3 GHz machine with 4 GB RAM

**** On a cluster of 32 64-bit machines with 1 GB RAM***

“Our load balance system crashes”

Case study for a printing service organization

Findings:

In 7,500 Lines of Code, 6 errors were found

With error traces, these errors were repaired

No further deadlocks have occurred up to now

"What is the quality of our driver?"

Case study for a chip manufacturer

Problem: ***"What is the quality of our driver?"***

(Focusing on dynamic properties)

Facts:

- **Driver shows race conditions**
- **Ca. 5,000 LoC in C language**
- **Driver for Linux Kernel**
- **Documentation did not give insight in problem issues**

“What is the quality of our driver?”

Case study for a chip manufacturer

- The source code was manually translated into a mathematical model describing the system’s behavior
- This model was checked fully automatically for behavioral properties such as:
  - Interrupt enabled accessing of shared memory
  - Disabling / Enabling interrupts twice in a row
  - Inconsistencies in use of wake-up functions
  - Incorrectly detected timeouts

“What is the quality of our driver?”

Case study for a chip manufacturer

Findings:

Mutual exclusion violations in accessing shared memory

Improper use of wake-up calls

Violations were traced back to the source code

Suggestions for fixes in the source code