Quality Assessments on Source Code at LaQuSo
What is LaQuSo? What do we do?
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 (??)
- code metrics
- dep. graph
- dupl. graph
- code smells / sloppiness
- coding standards
- arch. issues
- process models
- dynamic properties
- bugs (?)
- misc. issues

tools + scripts:
- analysis
- visualization

browsing ('expert review')
Software quality assessment
with *static*, not *dynamic* analysis

- source code
- comments
- running system (?)
- user/install docs (?)
- design docs (??)
- models (??)
- code metrics
- dep. graph
- dupl. graph
- code smells / sloppiness
- coding standards conformance
- arch. issues
- bugs (?)
- process models
- dynamic properties
- arch. issues
- bugs (?)
- process models
- misc. issues

*tools + scripts:*
- analysis
- visualization
- browsing (‘expert review’)
Software quality assessment
with static, *structural*, not *behavioral* analysis

- source code
- comments
- code metrics
- dep. graph
- dupl. graph
- user/install docs (?)
- design docs (??)
- models (??)
- code smells / sloppiness
- arch. issues
- coding standards
- coding conformance
- bugs (?)
- process models
- misc. issues
- dynamic properties

*tools + scripts:*
- analysis
- visualization

*LaQuSo*

*browsing* (*‘expert review’*)
Software quality assessment with static, *structural*, not *behavioral* analysis

**structure** ("architecture"): how is it put together?
- code quality metrics
- dependency graph analysis
- duplication graph analysis

**behavior**: what does it do?
- dataflow analysis
- assertion checking
- model checking
Software quality assessment
on just source code (design documentation is rare)

- source code
- comments

*tools + scripts:*
- analysis
- visualization

- code metrics
- dep. graph
- dupl. graph

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

- browsing
  ('expert review')

- ...
Source code quality assessment
with static, structural analysis

- source code
- comments

tools + scripts:
- analysis
- visualization

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

browsing (‘expert review’)

LaQuSo
Laboratory for Quality Software
Source code quality assessment
with static, structural analysis

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

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

$$\Delta E = E$$

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
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)

Layers

Callers

Called

LaQuSo
Laboratory for Quality Software
“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 File A

Code File B

Duplication between files

Internal duplication
“How maintainable is our system?”
Case study for a financial organization

• 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)
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 an embedded systems manufacturer

Question:  "With extensive changes to the system, is architectural purity still preserved?"
(Deployers 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 an embedded systems manufacturer

Findings:

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 9i 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

<table>
<thead>
<tr>
<th>Layer</th>
<th>Unit</th>
<th>Module</th>
<th>LOCs</th>
<th>comment</th>
<th>Blanks</th>
<th>Source</th>
<th>Ifs</th>
<th>LOOPS</th>
<th>McCabe</th>
<th>Fan_in</th>
<th>Fan_out</th>
<th>CLN</th>
<th>NBR</th>
<th>RSA</th>
<th>RSI</th>
<th>CVR</th>
<th>id</th>
</tr>
</thead>
<tbody>
<tr>
<td>CobolPr</td>
<td>XOFC</td>
<td>'XOFC27'</td>
<td>17480</td>
<td>2333</td>
<td>310</td>
<td>14837</td>
<td>256</td>
<td>63</td>
<td>320</td>
<td>0</td>
<td>21</td>
<td>768</td>
<td>8</td>
<td>0.2720</td>
<td>0.60300</td>
<td>0.71600</td>
<td></td>
</tr>
<tr>
<td>CobolPr</td>
<td>XOFC</td>
<td>'XOFC27'</td>
<td>16331</td>
<td>1627</td>
<td>463</td>
<td>14241</td>
<td>194</td>
<td>105</td>
<td>300</td>
<td>1</td>
<td>33</td>
<td>887</td>
<td>8</td>
<td>0.53000</td>
<td>0.59900</td>
<td>0.71700</td>
<td></td>
</tr>
<tr>
<td>CobolPr</td>
<td>XOFC</td>
<td>'XOFC27'</td>
<td>8722</td>
<td>707</td>
<td>313</td>
<td>7702</td>
<td>105</td>
<td>25</td>
<td>131</td>
<td>1</td>
<td>20</td>
<td>715</td>
<td>8</td>
<td>0.82900</td>
<td>0.51500</td>
<td>0.86100</td>
<td></td>
</tr>
<tr>
<td>CobolPr</td>
<td>XOFC</td>
<td>'XOFC27'</td>
<td>11391</td>
<td>911</td>
<td>598</td>
<td>9882</td>
<td>87</td>
<td>26</td>
<td>114</td>
<td>1</td>
<td>21</td>
<td>771</td>
<td>8</td>
<td>0.69600</td>
<td>0.49900</td>
<td>0.73000</td>
<td></td>
</tr>
<tr>
<td>CobolPr</td>
<td>XUFIC</td>
<td>'XUFIC6'</td>
<td>1688</td>
<td>249</td>
<td>104</td>
<td>1335</td>
<td>47</td>
<td>2</td>
<td>50</td>
<td>1</td>
<td>0</td>
<td>66</td>
<td>8</td>
<td>0.21700</td>
<td>0.19400</td>
<td>0.27200</td>
<td></td>
</tr>
<tr>
<td>CobolPr</td>
<td>XIFC</td>
<td>'XIFC6'</td>
<td>1100</td>
<td>147</td>
<td>65</td>
<td>888</td>
<td>33</td>
<td>5</td>
<td>39</td>
<td>1</td>
<td>5</td>
<td>25</td>
<td>8</td>
<td>0.15900</td>
<td>0.51700</td>
<td>0.59900</td>
<td></td>
</tr>
<tr>
<td>CobolPr</td>
<td>XIFC</td>
<td>'XIFC6'</td>
<td>3000</td>
<td>185</td>
<td>199</td>
<td>2616</td>
<td>11</td>
<td>12</td>
<td>24</td>
<td>1</td>
<td>7</td>
<td>176</td>
<td>8</td>
<td>0.27600</td>
<td>0.51400</td>
<td>0.66200</td>
<td></td>
</tr>
<tr>
<td>Rekenre</td>
<td>XARF</td>
<td>'XARFO31061744'</td>
<td>49</td>
<td>31</td>
<td>304</td>
<td>10</td>
<td>0</td>
<td>11</td>
<td>1</td>
<td>0</td>
<td>7</td>
<td>4</td>
<td>0.09800</td>
<td>0.34300</td>
<td>0.44000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Datalav</td>
<td>'XU06'</td>
<td>'XU0610671'</td>
<td>106</td>
<td>56</td>
<td>509</td>
<td>8</td>
<td>0</td>
<td>9</td>
<td>1</td>
<td>0</td>
<td>4</td>
<td>2</td>
<td>0.02300</td>
<td>0.63000</td>
<td>0.64500</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CobolPr</td>
<td>XNFC</td>
<td>'XNFC6'</td>
<td>1112</td>
<td>133</td>
<td>143</td>
<td>836</td>
<td>6</td>
<td>0</td>
<td>7</td>
<td>1</td>
<td>6</td>
<td>115</td>
<td>8</td>
<td>0.54300</td>
<td>0.40900</td>
<td>0.56400</td>
<td></td>
</tr>
<tr>
<td>CobolPr</td>
<td>XNFC</td>
<td>'XNFC6'</td>
<td>1300</td>
<td>109</td>
<td>52</td>
<td>469</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>1</td>
<td>8</td>
<td>48</td>
<td>8</td>
<td>0.37900</td>
<td>0.16700</td>
<td>0.37900</td>
<td></td>
</tr>
<tr>
<td>Datalav</td>
<td>'XU06'</td>
<td>'XU061061200'</td>
<td>96</td>
<td>34</td>
<td>110</td>
<td>2</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>0.13900</td>
<td>0.00000</td>
<td>0.13900</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rekenre</td>
<td>XARF</td>
<td>'XARFO3100130'</td>
<td>10</td>
<td>28</td>
<td>92</td>
<td>2</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>4</td>
<td>0.48900</td>
<td>0.00000</td>
<td>0.49900</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Datalav</td>
<td>'XU06'</td>
<td>'XU06106177'</td>
<td>41</td>
<td>33</td>
<td>103</td>
<td>2</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>0.13900</td>
<td>0.00000</td>
<td>0.13900</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rekenre</td>
<td>XARF</td>
<td>'XARFO3119'</td>
<td>10</td>
<td>28</td>
<td>81</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>7</td>
<td>4</td>
<td>0.94800</td>
<td>0.00000</td>
<td>0.94800</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rekenre</td>
<td>XARF</td>
<td>'XARFO3117'</td>
<td>10</td>
<td>28</td>
<td>79</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>4</td>
<td>0.59800</td>
<td>0.00000</td>
<td>0.59800</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rekenre</td>
<td>XARF</td>
<td>'XARFO3119'</td>
<td>10</td>
<td>28</td>
<td>81</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>7</td>
<td>4</td>
<td>0.94300</td>
<td>0.00000</td>
<td>0.94300</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- McCabe complexity
  (#If’s + #Loops +1)

- Fan Out
  (# modules called)
Calculating quality metrics on the source code

**Guideline:** McCabe ≤ 30

Some are over 100 going up to 320!

This rules out white-box testing

**McCabe complexity**

(#If’s + #Loops +1)

Metrics can find maintenance landmines

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

Case study for a pension fund
“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!
Class Diagrams

<table>
<thead>
<tr>
<th>Metrics</th>
<th>Subsystems</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A</td>
</tr>
<tr>
<td>Number of classes</td>
<td>176</td>
</tr>
<tr>
<td>Number of methods</td>
<td>1106</td>
</tr>
<tr>
<td>Avg. methods per class</td>
<td>6.28</td>
</tr>
<tr>
<td>Classes with &gt; 30 methods</td>
<td>4</td>
</tr>
<tr>
<td>Max fan-in / Max fan-out</td>
<td>27 / 27</td>
</tr>
</tbody>
</table>

- 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

<table>
<thead>
<tr>
<th>Metrics</th>
<th>Subsystems</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A</td>
</tr>
<tr>
<td>Incoming and outgoing messages per class</td>
<td>Maximum</td>
</tr>
<tr>
<td></td>
<td>Classes with &gt; 30 mess.</td>
</tr>
<tr>
<td>Max. depth of scenario</td>
<td>41</td>
</tr>
</tbody>
</table>

- 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.
- ...

LaQuSo
Laboratory for Quality Software
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:
                    ... // code
                    break;
            }
        case STATE_B:
            switch (event) {
                case EVENT1:
                    ... // code
                    break;
                case EVENT3:
                    ... // code
                    break;
            }
        default: ... // code
    }
}
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
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

Client 1 → Load Balancer → Printer 1
Client 2 → Load Balancer → Printer 2
...
Client n → Load Balancer → Printer m
“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:

<table>
<thead>
<tr>
<th>#Clients</th>
<th>#Servers</th>
<th>Time</th>
<th>#Levels</th>
<th>#States</th>
<th>#Transitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>7m38s</td>
<td>241</td>
<td>657k</td>
<td>1.38M</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3h01m</td>
<td>267</td>
<td>18M</td>
<td>38.5M</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>9h55m</td>
<td>444</td>
<td>54M</td>
<td>141M</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>13h*</td>
<td>481</td>
<td>213M</td>
<td>465.5M</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>&gt;113h*</td>
<td>&gt;215</td>
<td>&gt;511M</td>
<td>&gt;1121M</td>
</tr>
</tbody>
</table>

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
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