How Green are Java Best Practices?

Best Coding Practices and Software Eco-Design

Jérôme Rocheteau

Institut Catholique d'Arts et Métiers, Nantes, France

GreenDays@Rennes – Mardi 1er juillet 2014
Overview

1. Eco-Design and Best Practices
2. Formalizing Practices
3. Measuring Codes
4. Analyzing Measures, Codes, Practices
5. Eco-Design Indicators
Eco-Design and Best Practices

1. Eco-Design and Best Practices
   - Context and Issues
   - Hypothesis and Objectives

2. Formalizing Practices

3. Measuring Codes

4. Analyzing Measures, Codes, Practices

5. Eco-Design Indicators
Context and Issues

Context:

- ICT accounted 2% of carbon emissions in 2007
- Energy efficiency relies on hardware but not software
- Works on energy efficiency classes, energy-aware systems

Issues:

- Help developers to build energy-efficient software
  1. Detect energy-consuming patterns in source code
  2. Replace these patterns by energy-saving ones
- Qualify the energy impact for such best practices
Hypothesis and Objectives

Hypothesis

Best coding practices are eco-design rules

Objectives:

1. Formalizing Java best practices
2. Measuring savings of memory and energy at runtime
3. Evaluating confidence of such results
Formalizing Practices

1 Eco-Design and Best Practices

2 Formalizing Practices
   - Informal and Formal Examples
   - Time, Space, Energy Semantics

3 Measuring Codes

4 Analyzing Measures, Codes, Practices

5 Eco-Design Indicators
Informal and Formal Examples

Best Coding Practice Example / Informal Rule:

String Literal Initialization
Initialization of strings with the keyword 'new' creates new objects. This is costly in time and memory space. Initializations with literal strings avoids this.
Informal and Formal Examples

Best Coding Practice Example / Informal Rule:

String Literal Initialization

Initialization of strings with the keyword ’new’ creates new objects. This is costly in time and memory space. Initializations with literal strings avoids this.

Three shape of Java best coding practices:

1. Prefer *this*
2. Avoid *that*
3. Replace *that* by *this*
Informal and Formal Examples

Best Coding Practice Example / Informal Rule:

String Literal Initialization
Initialization of strings with the keyword 'new' creates new objects. This is costly in time and memory space. Initializations with literal strings avoids this.

- Three shape of Java best coding practices:
  1. Prefer this
  2. Avoid that
  3. Replace that by this

- Can be extended to other primitive/wrapper types
Informal and Formal Examples

Best Coding Practice Example / Informal Rule:

String Literal Initialization
Initialization of strings with the keyword 'new' creates new objects. This is costly in time and memory space. Initializations with literal strings avoids this.

- Three shape of Java best coding practices:
  1. Prefer this
  2. Avoid that
  3. Replace that by this

- Can be extended to other primitive/wrapper types
- Can be applied to other programming languages
Informal and Formal Examples

Best Coding Practice Example / Informal Rule:

String Literal Initialization
Initialization of strings with the keyword 'new' creates new objects. This is costly in time and memory space. Initializations with literal strings avoids this.

- Three shape of Java best coding practices:
  1. Prefer *this*
  2. Avoid *that*
  3. Replace *that* by *this*

- Can be extended to other primitive/wrapper types
- Can be applied to other programming languages
Informal and Formal Examples

Listing 1: Prefer String Literal Initialization

     <rule id="prefer-string-literal-initialization">
     <title>Prefer string literal initialization</title>
     <description>
     Primitive type objects should be initialized with primitive values and without the use of any constructors.
     </description>
     <check green="StringValue" gray="StringObject" />
     </rule>
Informal and Formal Examples

Listing 2: String Literal Initialization

```java
class StringValue implements Code {

    private String[] array;

    public void setUp() {
        array = new String[1000];
    }

    public void doRun() throws Exception {
        for (int i = 0; i < 1000; i++) {
            array[i] = "abcdefg . . . ";
        }
    }

    public void tearDown() {
        array = null;
    }
}
```
Informal and Formal Examples

Listing 3: String Object Initialization

```java
public class StringObject implements Code {

    private String[] array;

    public void setUp() {
        array = new String[1000];
    }

    public void doRun() throws Exception {
        for (int i = 0; i < 1000; i++) {
            array[i] = new String("abcdefg... ");
        }
    }

    public void tearDown() {
        array = null;
    }

}
```
Prefer Literal Initialization

\[
\begin{align*}
time &= f_t(t_d, t_c) \\
\text{space} &= f_s(s_d, s_c) \\
\text{energy} &= f_e(e_d, e_c)
\end{align*}
\]
Time, Space, Energy Semantics

String Value Initialization

Prefer Literal Initialization

String Object Initialization

\[
\begin{align*}
time &= f_t(t_d, t_c) \\
space &= f_s(s_d, s_c) \\
energy &= f_e(e_d, e_c)
\end{align*}
\]
Time, Space, Energy Semantics

String Value Initialization

Prefer Literal Initialization

String Object Initialization

- time $t_c$
- space $s_c$
- energy $e_c$

- time $t_d$
- space $s_d$
- energy $e_d$

$\text{time} = f_t(t_d, t_c)$
$\text{space} = f_s(s_d, s_c)$
$\text{energy} = f_e(e_d, e_c)$
Time, Space, Energy Semantics

String Value Initialization

Prefer Literal Initialization

String Object Initialization

\[
\begin{align*}
time &= f_t(t_d, t_c) \\
\text{space} &= f_s(s_d, s_c) \\
\text{energy} &= f_e(e_d, e_c)
\end{align*}
\]
Goal: statistical static analysis method and tools
Goal: statistical static analysis method and tools
Goal: statistical static analysis method and tools
Goal: statistical static analysis method and tools

possible savings

$X \text{ joules} / X \%$
Measuring Codes

1. Eco-Design and Best Practices

2. Formalizing Practices

3. Measuring Codes
   - Needs and Requirements
   - Measure Task and Process

4. Analyzing Measures, Codes, Practices

5. Eco-Design Indicators
Needs and Requirements

Needs:

Requirements:
Needs and Requirements

Needs:

- power-meter with digital outputs

Requirements:

- power-meter with fine-grain precision
Needs and Requirements

Needs:

- power-meter with digital outputs
- memory monitor with digital outputs

Requirements:

- power-meter with fine-grain precision
Needs and Requirements

Needs:

- power-meter with digital outputs
- memory monitor with digital outputs
- platform for running Java codes

Requirements:

- power-meter with fine-grain precision
- Java micro-benchmarking to avoid JIT
Needs and Requirements

Needs:

- power-meter with digital outputs
- memory monitor with digital outputs
- platform for running Java codes
- system for managing codes and measures

Requirements:

- power-meter with fine-grain precision
- Java micro-benchmarking to avoid JIT
- system able to manage physical and logical sensors
Measure Task and Process

Measurement Protocol:

1. 4 seconds idle
2. 10 seconds execution time
3. 3 seconds idle
Measure Task and Process

Semantics based on quantitative metrics:

- **x-axis** execution time
- **y-axis** instant power or instant memory space
Measure Task and Process

Preliminary Computations:

- Idle power = average of the first 4 second instant powers
- Idle energy = idle power × protocol time
Measure Task and Process

Preliminary Computations:

- total energy obtained by the trapezoidal rule
Measure Task and Process

Preliminary Computations:

- total energy obtained by the trapezoidal rule
- idle power = average of the first 4 second instant powers
Measure Task and Process

**Preliminary Computations:**

- total energy obtained by the trapezoidal rule
- idle power = average of the first 4 second instant powers
- idle energy = idle power $\times$ protocol time
Measure Task and Process

Code energy computation and normalization:

- code energy = total energy - idle energy
Measure Task and Process

Code energy computation and normalization:

- code energy = total energy - idle energy
- normalized code energy = code energy / times of execution
Measure Task and Process

Measure Process:

1. Retrieve available Java code
2. Iterate while this code isn't mature:
   - Clean its measure set
   - Compute its maturity
   - Plan new measures if required
   - Perform these measures
3. Send the report of this code
Measure Task and Process

Measure Process:

1. Retrieve an available Java code
Measure Task and Process

Measure Process:

1. Retrieve an available Java code
2. Iterate while this code isn’t mature:
   - Clean its measure set
   - Compute its maturity
   - Plan new measures if required
   - Perform these measures
3. Send the report of this code
Measure Task and Process

Measure Process:

1. Retrieve an available Java code
2. Iterate while this code isn’t mature:
   - Clean its measure set
Measure Task and Process

Measure Process:

1. Retrieve an available Java code
2. Iterate while this code isn’t mature:
   - Clean its measure set
   - Compute its maturity
3. Send the report of this code
Measure Task and Process

Measure Process:

1. Retrieve an available Java code
2. Iterate while this code isn’t mature:
   - Clean its measure set
   - Compute its maturity
   - Plan new measures if required

3. Perform these measures

4. Send the report of this code
Measure Task and Process

Measure Process:

1. Retrieve an available Java code
2. Iterate while this code isn’t mature:
   - Clean its measure set
   - Compute its maturity
   - Plan new measures if required
   - Perform these measures
3. Send the report of this code
Analyzing Measures, Codes, Practices

1. Eco-Design and Best Practices

2. Formalizing Practices

3. Measuring Codes

4. Analyzing Measures, Codes, Practices
   - Clean and Canonical Measures
   - Code Maturity
   - Results

5. Eco-Design Indicators
Clean and Canonical Measures

3 kinds of measure disturbances:
Clean and Canonical Measures

3 kinds of measure disturbances:

- disturbances *before* the measure task $\leadsto$ underestimate
Clean and Canonical Measures

3 kinds of measure disturbances:

- disturbances *before* the measure task $\rightsquigarrow$ underestimate
- disturbances *after* the measure task $\rightsquigarrow$ overestimate
Clean and Canonical Measures

3 kinds of measure disturbances:

- disturbances *before* the measure task $\leadsto$ underestimate
- disturbances *after* the measure task $\leadsto$ overestimate
- disturbances *during* the measure task $\leadsto$ overestimate
Clean and Canonical Measures

3 kinds of measure disturbances:

- disturbances before the measure task
- disturbances after the measure task
- disturbances during the measure task

2 mere algorithms for cleaning measures:
Clean and Canonical Measures

3 kinds of measure disturbances:

- disturbances *before* the measure task
- disturbances *after* the measure task
- disturbances *during* the measure task

2 mere algorithms for cleaning measures:

- bounds checking algorithm (over a single measure)
Clean and Canonical Measures

3 kinds of measure disturbances:

- disturbances *before* the measure task
- disturbances *after* the measure task
- disturbances *during* the measure task

2 mere algorithms for cleaning measures:

- bounds checking algorithm (over a single measure)
- split-and-merge algorithm (over a set of measures)
Clean and Canonical Measures

Cleaning algorithm evaluation:

- 500 clean measures from 25 rules annotated by 3 experts
Clean and Canonical Measures

Cleaning algorithm evaluation:

- 500 clean measures from 25 rules annotated by 3 experts
- inter-agreement $\kappa$: 0.94 (almost perfect)
Clean and Canonical Measures

Cleaning algorithm evaluation:

- 500 clean measures from 25 rules annotated by 3 experts
- inter-agreement $\kappa$: 0.94 (almost perfect)
- baseline = quartile method
  - precision: 0.953
  - recall: 0.911
Clean and Canonical Measures

Cleaning algorithm evaluation:

- 500 clean measures from 25 rules annotated by 3 experts
- inter-agreement $\kappa$: 0.94 (almost perfect)
- baseline = quartile method
  - precision: 0.953
  - recall: 0.911
- split-and-merge algorithm
  - precision: 0.941
  - recall: 1.0
Clean and Canonical Measures

Cleaning algorithm evaluation:

- 500 clean measures from 25 rules annotated by 3 experts
- inter-agreement $\kappa$: 0.94 (almost perfect)
- baseline $=$ quartile method
  - precision: 0.953
  - recall: 0.911
- split-and-merge algorithm
  - precision: 0.941
  - recall: 1.0
  - remove all disturbed measures
Clean and Canonical Measures

Canonical measure:

[Graph showing data over time]
Clean and Canonical Measures

Canonical measure:

time: $t_1$

space: $s_1$

energy: $e_1$
Clean and Canonical Measures

Canonical measure:

time: $t_1$

space: $s_1$

energy: $e_1$

time: $t$

space: $s$

energy: $e$
Clean and Canonical Measures

Canonical measure:
Clean and Canonical Measures

Canonical measure:

- **time**: $t_1$
- **space**: $s_1$
- **energy**: $e_1$

- **time**: $t$
- **space**: $s$
- **energy**: $e$
Clean and Canonical Measures

Canonical measure:

\[
\begin{align*}
\text{time: } & t_1 \\
\text{space: } & s_1 \\
\text{energy: } & e_1
\end{align*}
\]

error margin: 2%

\[
\begin{align*}
\text{time: } & t \\
\text{space: } & s \\
\text{energy: } & e
\end{align*}
\]
Code Maturity

How many clean measures required for reliable results?
Code Maturity

How many clean measures required for reliable results?

How Green are Java Best Practices? GreenDays | 2014-07-01
Code Maturity

How many clean measures required for reliable results?

- a set of 25 measures minimum
Code Maturity

How many clean measures required for reliable results?

- a set of 25 measures minimum
- a standard deviation less than 10%
Energy in nanojoules, time in nanoseconds
Memory in kilobytes
Standard Deviation on Energy

Replace Object Initialization by Literal Initialization

<table>
<thead>
<tr>
<th>Rule Id</th>
<th>Green Energy</th>
<th>Gray Energy</th>
<th>Green Time</th>
<th>Gray Time</th>
<th>Green Memory</th>
<th>Gray Memory</th>
<th>Green StdDev</th>
<th>Gray StdDev</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>697</td>
<td>7885</td>
<td>842</td>
<td>7827</td>
<td>4136</td>
<td>36104</td>
<td>4.40%</td>
<td>8.14%</td>
</tr>
<tr>
<td>Float</td>
<td>10311</td>
<td>10448</td>
<td>4736</td>
<td>4127</td>
<td>20032</td>
<td>20032</td>
<td>5.85%</td>
<td>6.58%</td>
</tr>
<tr>
<td>Integer</td>
<td>685</td>
<td>9575</td>
<td>833</td>
<td>4591</td>
<td>4048</td>
<td>20032</td>
<td>5.21%</td>
<td>6.51%</td>
</tr>
<tr>
<td>Boolean</td>
<td>683</td>
<td>6267</td>
<td>775</td>
<td>4741</td>
<td>4048</td>
<td>20032</td>
<td>4.77%</td>
<td>7.03%</td>
</tr>
<tr>
<td>Char</td>
<td>695</td>
<td>33067</td>
<td>840</td>
<td>4595</td>
<td>4048</td>
<td>20032</td>
<td>5.04%</td>
<td>4.65%</td>
</tr>
<tr>
<td>Double</td>
<td>10003</td>
<td>10210</td>
<td>5810</td>
<td>4270</td>
<td>28032</td>
<td>28032</td>
<td>5.58%</td>
<td>7.83%</td>
</tr>
<tr>
<td>Long</td>
<td>669</td>
<td>8236</td>
<td>807</td>
<td>6066</td>
<td>4056</td>
<td>28032</td>
<td>2.89%</td>
<td>5.32%</td>
</tr>
<tr>
<td>Short</td>
<td>680</td>
<td>7819</td>
<td>819</td>
<td>3846</td>
<td>4048</td>
<td>20031</td>
<td>4.87%</td>
<td>7.71%</td>
</tr>
</tbody>
</table>
## Results

### Replace Object Initialization by Literal Initialization

<table>
<thead>
<tr>
<th>Rule Id</th>
<th>Green Energy</th>
<th>Gray Energy</th>
<th>Absolute Gain</th>
<th>Relative Gain</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>697</td>
<td>7885</td>
<td>7188</td>
<td>91.16%</td>
</tr>
<tr>
<td>Float</td>
<td>10311</td>
<td>10448</td>
<td>137</td>
<td>1.31%</td>
</tr>
<tr>
<td>Integer</td>
<td>685</td>
<td>9575</td>
<td>8890</td>
<td>92.54%</td>
</tr>
<tr>
<td>Boolean</td>
<td>683</td>
<td>6267</td>
<td>5584</td>
<td>89.10%</td>
</tr>
<tr>
<td>Char</td>
<td>695</td>
<td>33067</td>
<td>32372</td>
<td>97.89%</td>
</tr>
<tr>
<td>Double</td>
<td>10003</td>
<td>10210</td>
<td>207</td>
<td>2.02%</td>
</tr>
<tr>
<td>Long</td>
<td>669</td>
<td>8236</td>
<td>7567</td>
<td>91.87%</td>
</tr>
<tr>
<td>Short</td>
<td>680</td>
<td>7819</td>
<td>7139</td>
<td>91.30%</td>
</tr>
</tbody>
</table>
Results

Rules for loops

A Prefer integer loop counters
B Avoid method loop conditions
C Prefer comparison-to-0 conditions
D Prefer first common condition

<table>
<thead>
<tr>
<th>Id</th>
<th>Green Energy</th>
<th>Gray Energy</th>
<th>Green Time</th>
<th>Gray Time</th>
<th>Green Memory</th>
<th>Gray Memory</th>
<th>Green StdDev</th>
<th>Gray StdDev</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>82</td>
<td>98</td>
<td>4744</td>
<td>5931</td>
<td>16</td>
<td>16</td>
<td>8.66%</td>
<td>7.46%</td>
</tr>
<tr>
<td>B</td>
<td>8376</td>
<td>8602</td>
<td>6181</td>
<td>6364</td>
<td>20056</td>
<td>20056</td>
<td>3.83%</td>
<td>6.14%</td>
</tr>
<tr>
<td>C</td>
<td>89</td>
<td>284</td>
<td>4709</td>
<td>17367</td>
<td>16</td>
<td>16</td>
<td>5.09%</td>
<td>5.78%</td>
</tr>
<tr>
<td>D</td>
<td>97</td>
<td>100</td>
<td>4934</td>
<td>5017</td>
<td>16</td>
<td>16</td>
<td>4.37%</td>
<td>4.60%</td>
</tr>
</tbody>
</table>
Rules for loops

A Prefer integer loop counters
B Avoid method loop conditions
C Prefer comparison-to-0 conditions
D Prefer first common condition

<table>
<thead>
<tr>
<th>Id</th>
<th>Green Energy</th>
<th>Gray Energy</th>
<th>Absolute Gain</th>
<th>Relative Gain</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>82</td>
<td>98</td>
<td>16</td>
<td>16.32%</td>
</tr>
<tr>
<td>B</td>
<td>8376</td>
<td>8602</td>
<td>226</td>
<td>2.62%</td>
</tr>
<tr>
<td>C</td>
<td>89</td>
<td>284</td>
<td>195</td>
<td>68.66%</td>
</tr>
<tr>
<td>D</td>
<td>97</td>
<td>100</td>
<td>3</td>
<td>3.00%</td>
</tr>
</tbody>
</table>
## Results

### Set String Builder or Buffer Size

<table>
<thead>
<tr>
<th>Id</th>
<th>Green Energy</th>
<th>Gray Energy</th>
<th>Green Time</th>
<th>Gray Time</th>
<th>Green Memory</th>
<th>Gray Memory</th>
<th>Green StdDev</th>
<th>Gray StdDev</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>593</td>
<td>1053</td>
<td>38085</td>
<td>59111</td>
<td>52056</td>
<td>73784</td>
<td>7.11%</td>
<td>8.40%</td>
</tr>
<tr>
<td>B</td>
<td>598</td>
<td>1053</td>
<td>38495</td>
<td>59111</td>
<td>52056</td>
<td>73784</td>
<td>7.86%</td>
<td>8.40%</td>
</tr>
<tr>
<td>C</td>
<td>1211</td>
<td>1053</td>
<td>59111</td>
<td>70765</td>
<td>73784</td>
<td>104064</td>
<td>8.40%</td>
<td>9.40%</td>
</tr>
<tr>
<td>A’</td>
<td>378</td>
<td>899</td>
<td>19278</td>
<td>41088</td>
<td>52056</td>
<td>73784</td>
<td>8.27%</td>
<td>7.18%</td>
</tr>
<tr>
<td>B’</td>
<td>429</td>
<td>899</td>
<td>41088</td>
<td>51214</td>
<td>73784</td>
<td>104064</td>
<td>7.18%</td>
<td>5.84%</td>
</tr>
<tr>
<td>C’</td>
<td>1043</td>
<td>899</td>
<td>20976</td>
<td>41088</td>
<td>52056</td>
<td>73784</td>
<td>6.01%</td>
<td>7.18%</td>
</tr>
</tbody>
</table>

How Green are Java Best Practices?
Results

Set String Builder or Buffer Size

A String Buffer/Builder capacity initialization
B String Buffer/Builder capacity setting
C String Buffer/Builder length setting

<table>
<thead>
<tr>
<th>Id</th>
<th>Green Energy</th>
<th>Gray Energy</th>
<th>Absolute Gain</th>
<th>Relative Gain</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>593</td>
<td>1053</td>
<td>460</td>
<td>43.68%</td>
</tr>
<tr>
<td>B</td>
<td>598</td>
<td>1053</td>
<td>455</td>
<td>43.20%</td>
</tr>
<tr>
<td>C</td>
<td>1211</td>
<td>1053</td>
<td>-158</td>
<td>-15.00%</td>
</tr>
<tr>
<td>A'</td>
<td>378</td>
<td>899</td>
<td>521</td>
<td>57.95%</td>
</tr>
<tr>
<td>B'</td>
<td>429</td>
<td>899</td>
<td>470</td>
<td>52.28%</td>
</tr>
<tr>
<td>C'</td>
<td>1043</td>
<td>899</td>
<td>-144</td>
<td>-16.01%</td>
</tr>
</tbody>
</table>
Eco-Design Indicators

1. Eco-Design and Best Practices

2. Formalizing Practices

3. Measuring Codes

4. Analyzing Measures, Codes, Practices

5. Eco-Design Indicators
   - Summary
   - Future
Eco-Design Indicators

Java (80)

HTML (24)

C/C++ (13)

JavaScript (20)

PHP (13)

HTTPD (5)

CSS (7)

SQL

Total (174)
Hypothesis

Best Coding Practices ≈ Eco-Design Rules

174 formalized and measured rules
Eco-Design Indicators

Hypothesis

Best Coding Practices \approx\ Eco-Design Rules

174 formalized and measured rules

- 55% rules \implies huge savings (more than 10%)
Eco-Design Indicators

Hypothesis

Best Coding Practices $\approx$ Eco-Design Rules

174 formalized and measured rules

- 55% rules $\Rightarrow$ huge savings (more than 10%)
- 20% rules $\Rightarrow$ few savings (between 3% and 10%)
Eco-Design Indicators

Hypothesis

Best Coding Practices \( \approx \) Eco-Design Rules

174 formalized and measured rules

- 55% rules \( \leadsto \) huge savings (more than 10%)
- 20% rules \( \leadsto \) few savings (between 3% and 10%)
- 15% rules \( \leadsto \) no savings (between -3% and 3%)
- 10% rules \( \leadsto \) some losses (less than -3%)
Eco-Design Indicators

Hypothesis

Best Coding Practices $\approx$ Eco-Design Rules

174 formalized and measured rules

- 55% rules $\leadsto$ huge savings (more than 10%)
- 20% rules $\leadsto$ few savings (between 3% and 10%)
- 15% rules $\leadsto$ no savings (between -3% and 3%)
- 10% rules $\leadsto$ some losses (less than -3%)
Eco-Design Indicators

With The Way: 1 reliable measure system
Eco-Design Indicators

With The Way: 1 reliable measure system

*hybrid* physical and logical sensor management
Eco-Design Indicators

With The Way: 1 reliable measure system

- **hybrid** physical and logical sensor management
- **complex** client/server architecture
Eco-Design Indicators

With The Way: 1 reliable measure system

- **hybrid** physical and logical sensor management
- **complex** client/server architecture
- **focused** star schema database
Eco-Design Indicators

With The Way: 1 reliable measure system

- **hybrid**  physical and logical sensor management
- **complex**  client/server architecture
- **focused**  star schema database
- **robust**  automatic iterative process
Eco-Design Indicators

With The Way: 1 reliable measure system

- **hybrid** physical and logical sensor management
- **complex** client/server architecture
- **focused** star schema database
- **robust** automatic iterative process
- **extensible** programming API
Eco-Design Indicators

With The Way: 1 reliable measure system

- **hybrid** physical and logical sensor management
- **complex** client/server architecture
- **focused** star schema database
- **robust** automatic iterative process
- **extensible** programming API
- **reliable** stable measure sets with few measures
Eco-Design Indicators

Future

1. Energy Efficiency Classes?
Future

1. Energy Efficiency Classes?
2. Eco-Design Indicators?
Future

1. Energy Efficiency Classes?
2. Eco-Design Indicators?
   - Absolute savings
Future

1 Energy Efficiency Classes?
2 Eco-Design Indicators?
   - Absolute savings
   - Relative savings
Future

1. Energy Efficiency Classes?
2. Eco-Design Indicators?
   - Absolute savings
   - Relative savings
3. Rule indicators in different environments
Future

1. Energy Efficiency Classes?
2. Eco-Design Indicators?
   - Absolute savings
   - Relative savings
3. Rule indicators in different environments
4. Rule indicators in different programming languages
Future

1. Energy Efficiency Classes?
2. Eco-Design Indicators?
   - Absolute savings
   - Relative savings
3. Rule indicators in different environments
4. Rule indicators in different programming languages
5. Energy efficiency of different programming languages
Future

1. Energy Efficiency Classes?
2. Eco-Design Indicators?
   - Absolute savings
   - Relative savings
3. Rule indicators in different environments
4. Rule indicators in different programming languages
5. Energy efficiency of different programming languages
6. Accuracy of different physical and/or logical sensors
Future

1. Energy Efficiency Classes?
2. Eco-Design Indicators?
   - Absolute savings
   - Relative savings
3. Rule indicators in different environments
4. Rule indicators in different programming languages
5. Energy efficiency of different programming languages
6. Accuracy of different physical and/or logical sensors
7. Energy and memory footprints of logical sensors
Thank you