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

Eco-Design and Best Practices

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

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

```xml
<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

```
public 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

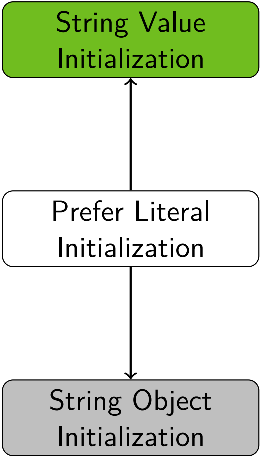
```
public class StringObject implements Code {

  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;
  }

}
```

Time, Space, Energy Semantics

Prefer Literal Initialization $\longrightarrow$

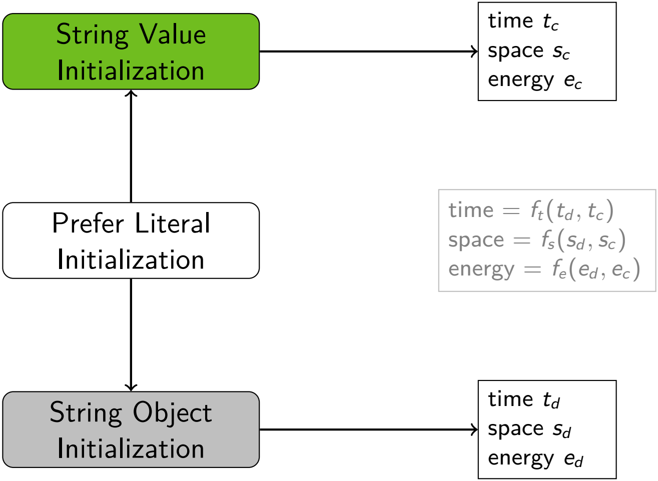
$$\begin{aligned} \text{time} &= f_t(t_d, t_c) \\ \text{space} &= f_s(s_d, s_c) \\ \text{energy} &= f_e(e_d, e_c) \end{aligned}$$

Time, Space, Energy Semantics

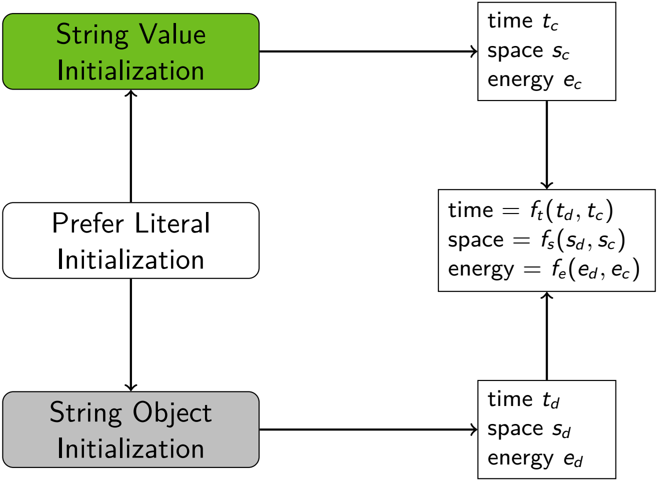


$$\text{time} = f_t(t_d, t_c)$$
$$\text{space} = f_s(s_d, s_c)$$
$$\text{energy} = f_e(e_d, e_c)$$

Time, Space, Energy Semantics



Time, Space, Energy Semantics



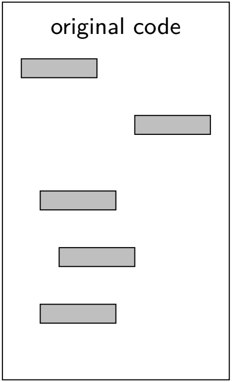
Time, Space, Energy Semantics

Goal: statistical static analysis method and tools

original code

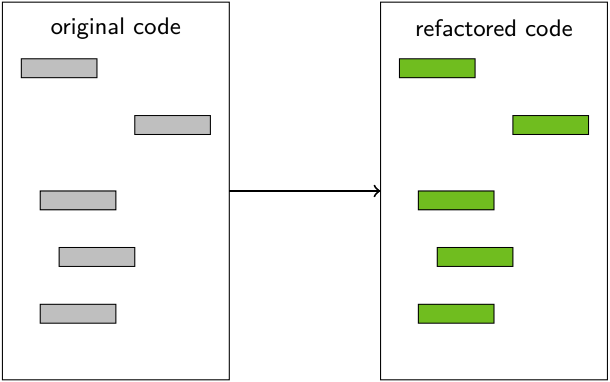
Time, Space, Energy Semantics

Goal: statistical static analysis method and tools



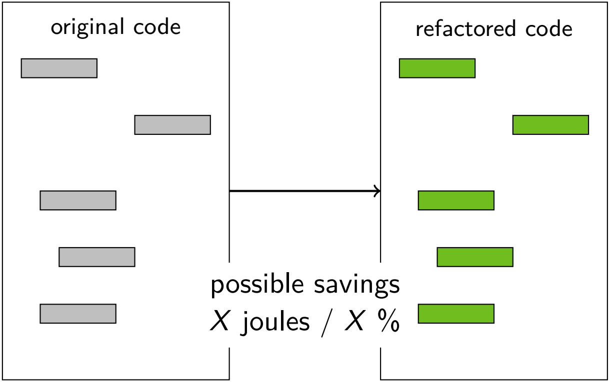
Time, Space, Energy Semantics

Goal: statistical static analysis method and tools



Time, Space, Energy Semantics

Goal: statistical static analysis method and tools



Measuring Codes

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

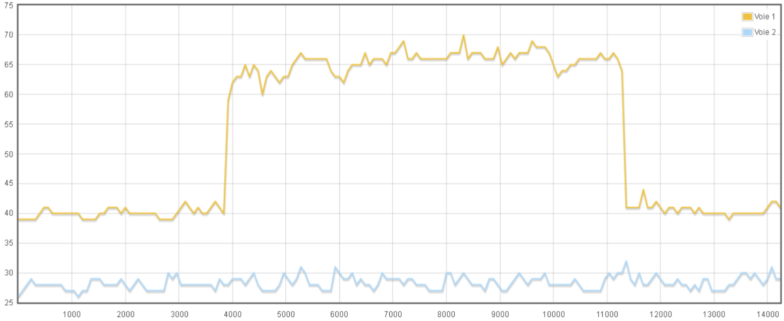
Requirements:

- 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:

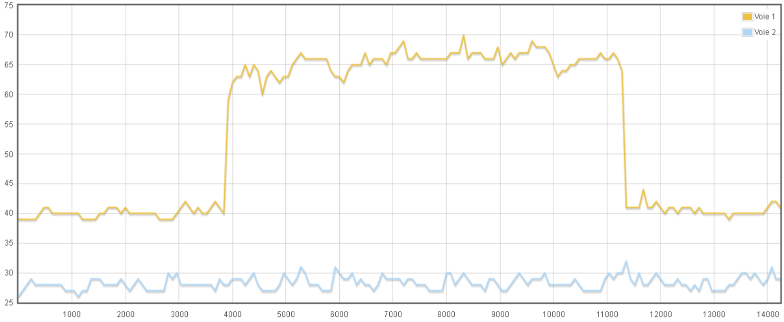
1. 4 seconds idle
2. 10 seconds execution time
3. 3 seconds idle



Measure Task and Process

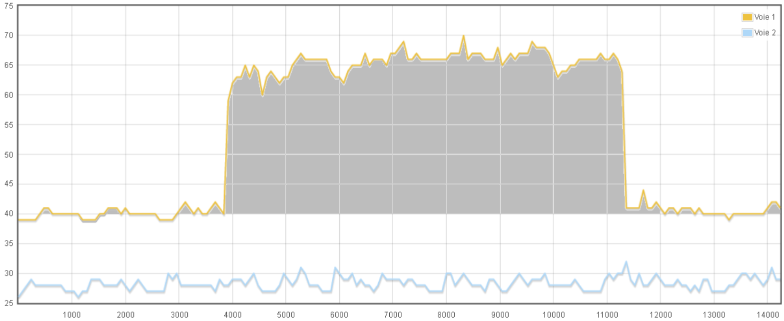
Semantics based on quantitative metrics:

- **x-axis** execution time
- **y-axis** instant power or instant memory space



Measure Task and Process

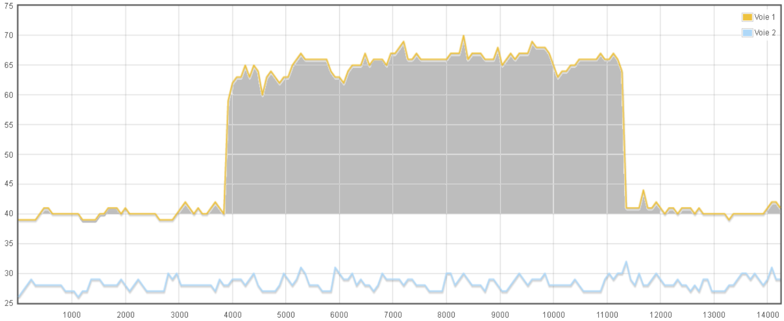
Preliminary Computations:



Measure Task and Process

Preliminary Computations:

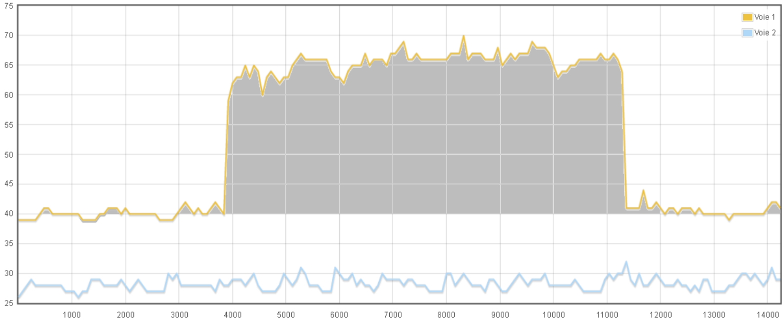
- total energy obtained by the trapezoidal rule



Measure Task and Process

Preliminary Computations:

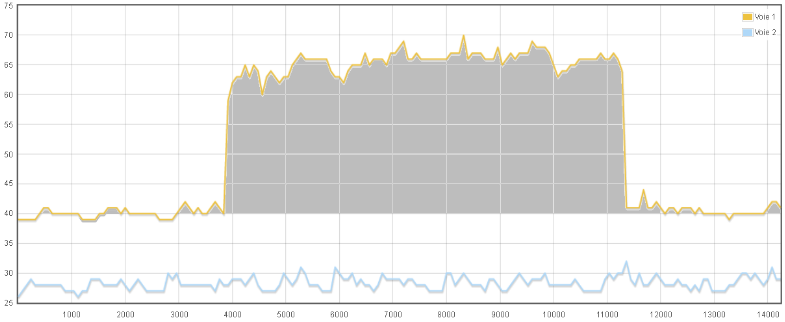
- total energy obtained by the trapezoidal rule
- idle power = average of the first 4 second instant powers



Measure Task and Process

Preliminary Computations:

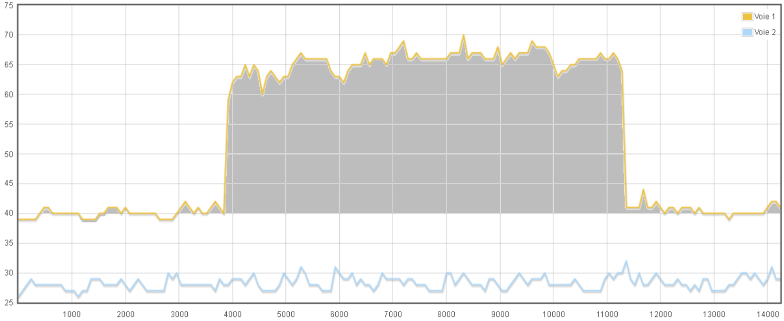
- total energy obtained by the trapezoidal rule
- idle power = average of the first 4 second instant powers
- idle energy = idle power × protocol time



Measure Task and Process

Code energy computation and normalization:

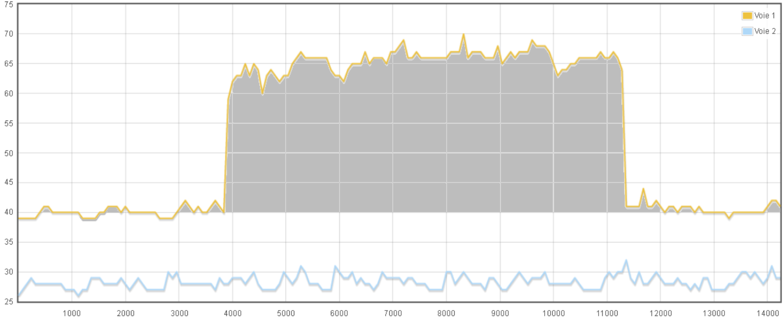
- code energy = total energy - idle energy



Measure Task and Process

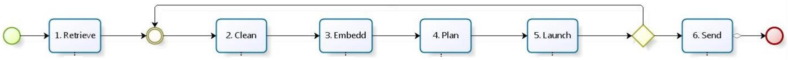
Code energy computation and normalization:

- code energy = total energy - idle energy
- normalized code energy = code energy / times of execution



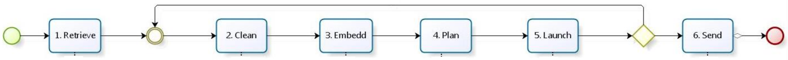
Measure Task and Process

Measure Process:



Measure Task and Process

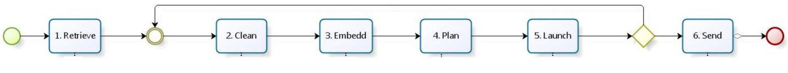
Measure Process:



1. Retrieve an available Java code

Measure Task and Process

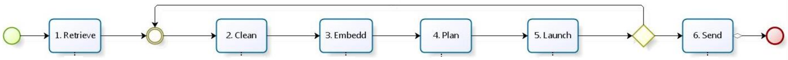
Measure Process:



1. Retrieve an available Java code
2. Iterate while this code isn't mature:

Measure Task and Process

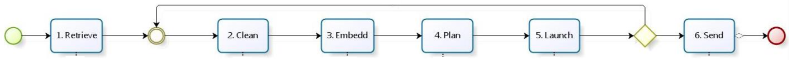
Measure Process:



1. Retrieve an available Java code
2. Iterate while this code isn't mature:
   - Clean its measure set

Measure Task and Process

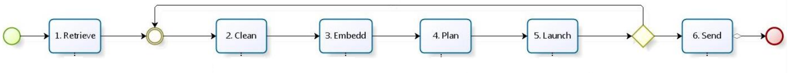
Measure Process:



1. Retrieve an available Java code
2. Iterate while this code isn't mature:
    - Clean its measure set
    - Compute its maturity

Measure Task and Process

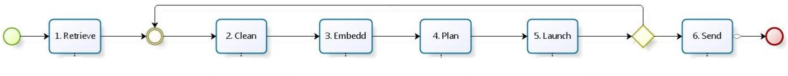
Measure Process:



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

Measure Task and Process

Measure Process:

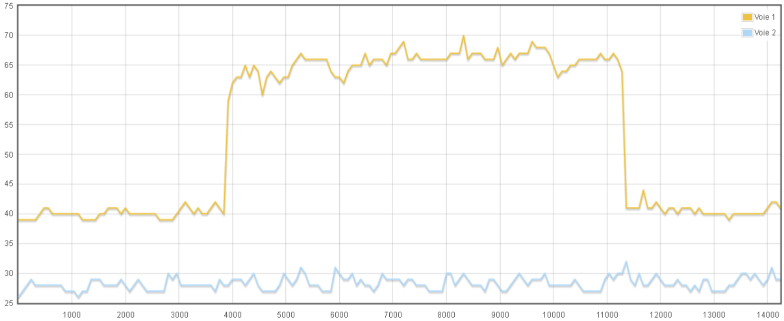


1. Retrieve an available Java code
2. Iterate while this code isn't mature:
   - Clean its measure set
   - Compute its maturity
   - Plan new measures if required
   - Perform these measures
3. Send the report of this code

Analyzing Measures, Codes, Practices

Clean and Canonical Measures

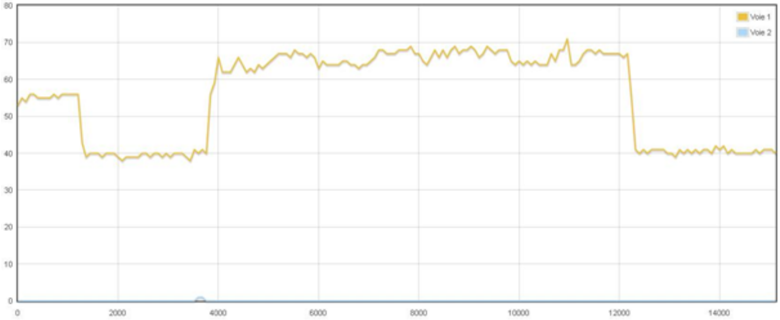
3 kinds of measure disturbances:



Clean and Canonical Measures

3 kinds of measure disturbances:

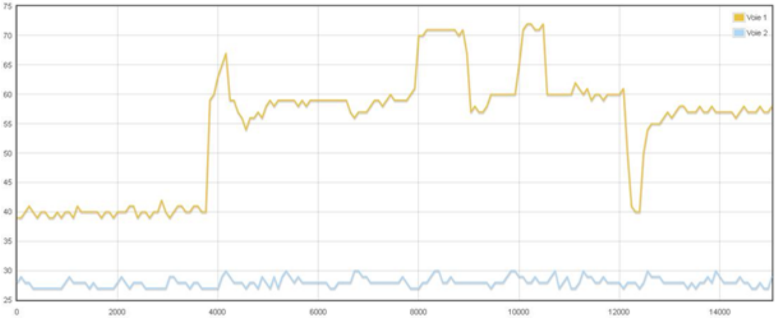
- disturbances *before* the measure task ⇝ underestimate



Clean and Canonical Measures

3 kinds of measure disturbances:

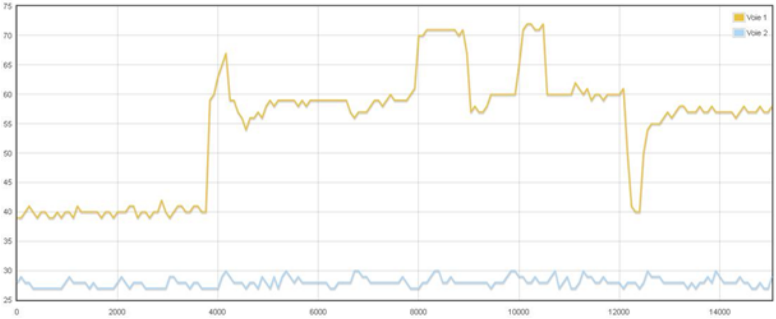
- disturbances *before* the measure task $\rightsquigarrow$ underestimate
- disturbances *after* the measure task $\rightsquigarrow$ overestimate



Clean and Canonical Measures

3 kinds of measure disturbances:

- disturbances *before* the measure task $\rightsquigarrow$ underestimate
- disturbances *after* the measure task $\rightsquigarrow$ overestimate
- disturbances *during* the measure task $\rightsquigarrow$ 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 κ: 0.94 (almost perfect)

Clean and Canonical Measures

Cleaning algorithm evaluation:

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

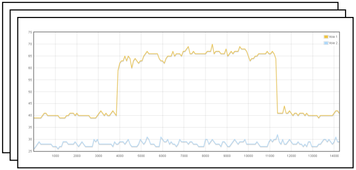
Clean and Canonical Measures

Cleaning algorithm evaluation:

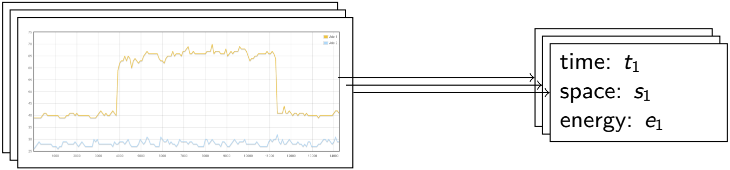
- 500 clean measures from 25 rules annotated by 3 experts
- inter-agreement κ: 0.94 (almost perfect)
- baseline = quartile method
  - precision: 0.953
  - recall: 0.911
- split-and-merge algorithm
  - precision: 0.941
  - recall: 1.0
  - **remove all disturbed measures**

Clean and Canonical Measures

Canonical measure:

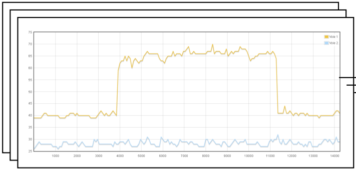


Canonical measure:



Clean and Canonical Measures

Canonical measure:

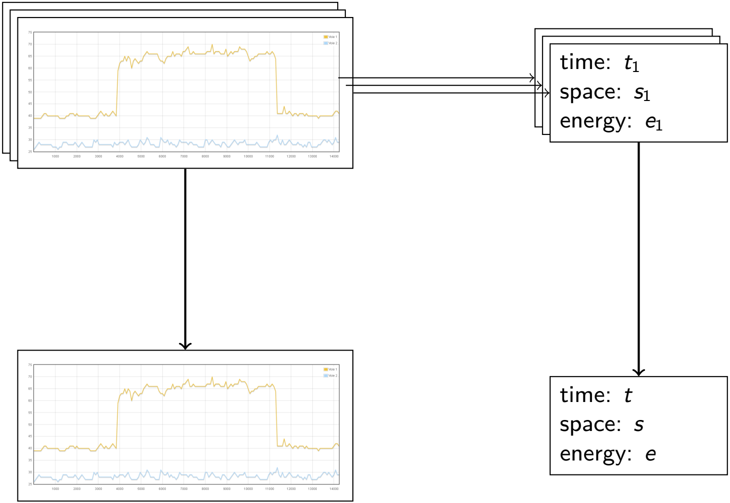


time: t_1
space: s_1
energy: e_1

time: t
space: s
energy: e

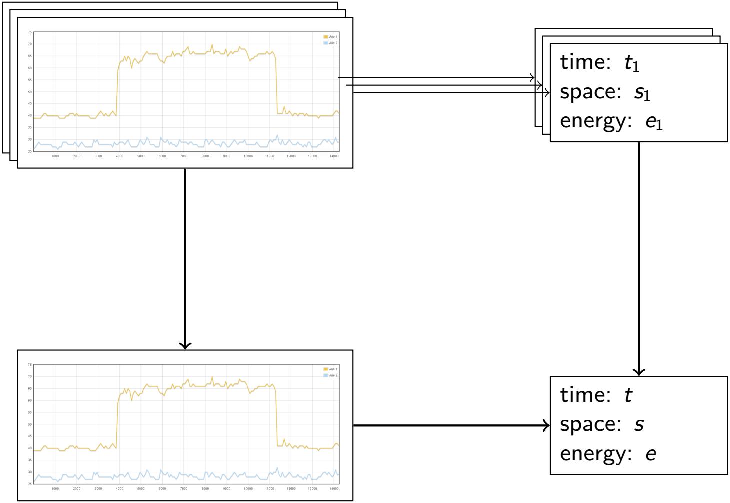
Clean and Canonical Measures

Canonical measure:



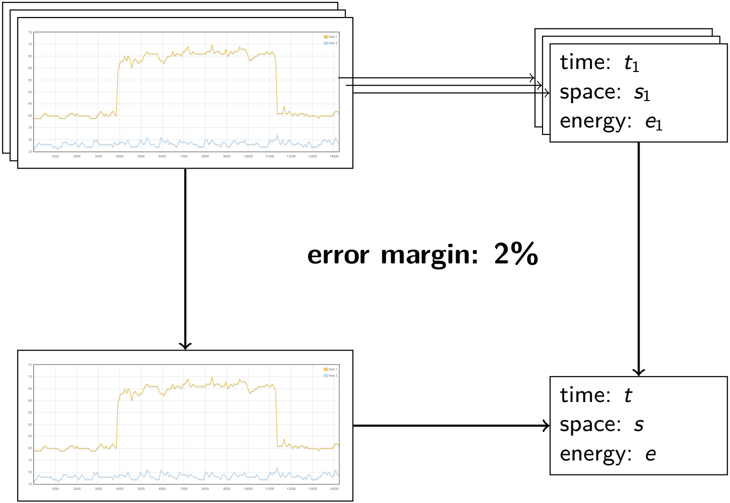
Clean and Canonical Measures

Canonical measure:



Clean and Canonical Measures

Canonical measure:



Code Maturity

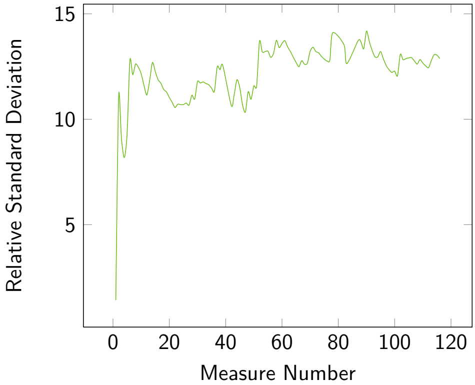
Code Maturity

How many clean measures required for reliable results?

Code Maturity

Code Maturity

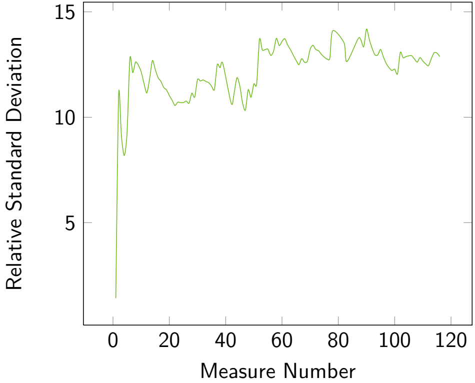
How many clean measures required for reliable results?



Code Maturity

Code Maturity

How many clean measures required for reliable results?

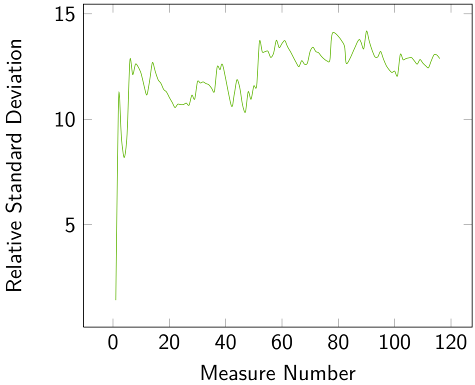


- a set of 25 measures minimum

Code Maturity

Code Maturity

How many clean measures required for reliable results?



- a set of 25 measures minimum
- a standard deviation less than 10%

Results

- Energy in nanojoules, time in nanoseconds
- Memory in kilobytes
- Standard Deviation on Energy

Replace Object Initialization by Literal Initialization

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

Results

Replace Object Initialization by Literal Initialization

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

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

Id	Green Energy	Gray Energy	Green Time	Gray Time	Green Memory	Gray Memory	Green StdDev	Gray StdDev
A	82	98	4744	5931	16	16	8.66%	7.46%
B	8376	8602	6181	6364	20056	20056	3.83%	6.14%
C	89	284	4709	17367	16	16	5.09%	5.78%
D	97	100	4934	5017	16	16	4.37%	4.60%

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

Id	Green Energy	Gray Energy	Absolute Gain	Relative Gain
A	82	98	16	16.32%
B	8376	8602	226	2.62%
C	89	284	195	68.66%
D	97	100	3	3.00%

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

Id	Green Energy	Gray Energy	Green Time	Gray Time	Green Memory	Gray Memory	Green StdDev	Gray StdDev
A	593	1053	38085	59111	52056	73784	7.11%	8.40%
B	598	1053	38495	59111	52056	73784	7.86%	8.40%
C	1211	1053	59111	70765	73784	104064	8.40%	9.40%
A'	378	899	19278	41088	52056	73784	8.27%	7.18%
B'	429	899	41088	51214	73784	104064	7.18%	5.84%
C'	1043	899	20976	41088	52056	73784	6.01%	7.18%

Results

Set String Builder or Buffer Size

A String Buffer/Builder capacity initialization

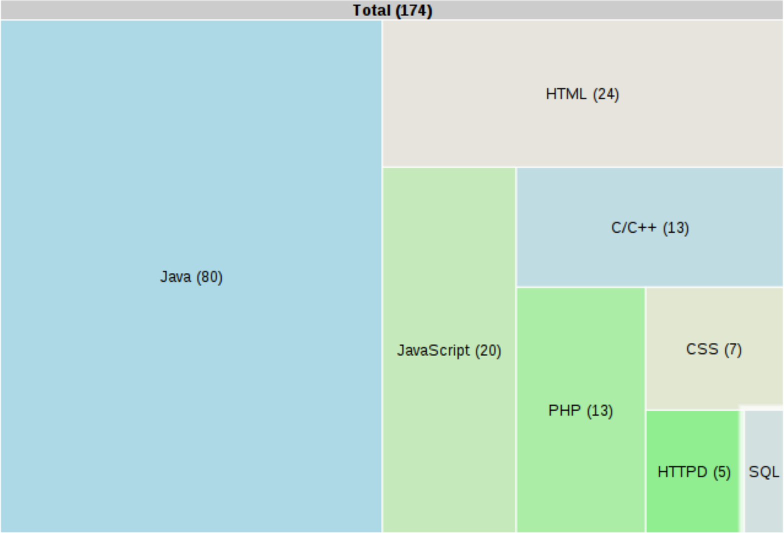
B String Buffer/Builder capacity setting

C String Buffer/Builder length setting

Id	Green Energy	Gray Energy	Absolute Gain	Relative Gain
A	593	1053	460	43.68%
B	598	1053	455	43.20%
C	1211	1053	-158	-15.00%
A'	378	899	521	57.95%
B'	429	899	470	52.28%
C'	1043	899	-144	-16.01%

Eco-Design Indicators

Eco-Design Indicators



Eco-Design Indicators

Hypothesis

Best Coding Practices $\approx$ 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 $\leadsto$ huge savings (more than 10%)

Eco-Design Indicators

Hypothesis

Best Coding Practices $\approx$ Eco-Design Rules

174 formalized and measured rules

- 55% rules $\rightsquigarrow$ huge savings (more than 10%)
- 20% rules $\rightsquigarrow$ few savings (between 3% and 10%)

Eco-Design Indicators

Hypothesis

Best Coding Practices $\approx$ Eco-Design Rules

174 formalized and measured rules

- 55% rules $\rightsquigarrow$ huge savings (more than 10%)
- 20% rules $\rightsquigarrow$ few savings (between 3% and 10%)
- 15% rules $\rightsquigarrow$ no savings (between -3% and 3%)

Eco-Design Indicators

Hypothesis

Best Coding Practices $\approx$ Eco-Design Rules

174 formalized and measured rules

- 55% rules $\rightsquigarrow$ huge savings (more than 10%)
- 20% rules $\rightsquigarrow$ few savings (between 3% and 10%)
- 15% rules $\rightsquigarrow$ no savings (between -3% and 3%)
- 10% rules $\rightsquigarrow$ 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

- Jérôme Rocheteau, Virginie Gaillard, et Lamya Belhaj.
  How Green are Java Best Coding Practices?
  Proceedings of the 3rd International Conference on Smart Grids and Green IT Systems,
  pages 235–246.
  Barcelona, Espagne, Avril 2014.

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

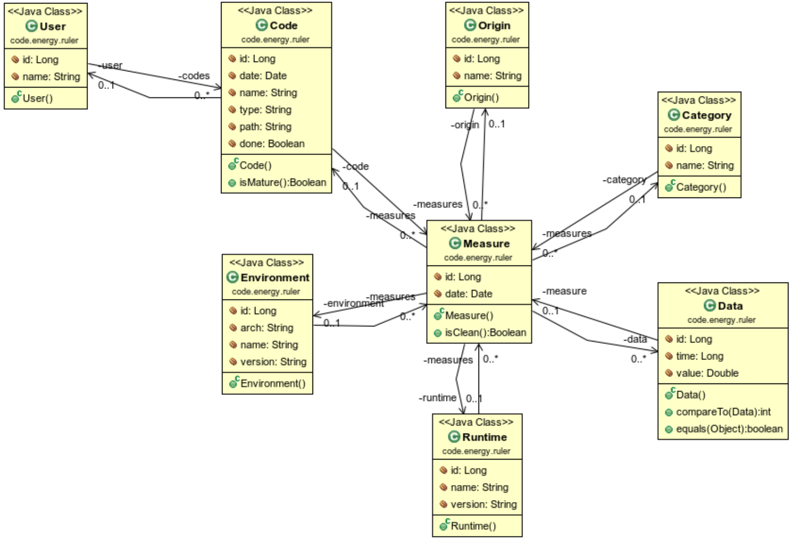
1. Energy Efficiency Classes?
2. Eco-Design Indicators?
   - Absolute savings
   - Relative savings
3. Rule indicators in different environments
4. Rule indicators in different programming languages
5. Energy efficiency of different programming languages
6. Accuracy of different physical and/or logical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

Data Model



<<Java Class>>
User
code.energy.ruler
id: Long
name: String
User()
<<Java Class>>
Code
code.energy.ruler
id: Long
date: Date
name: String
type: String
path: String
done: Boolean
Code()
isMature():Boolean
<<Java Class>>
Origin
code.energy.ruler
id: Long
name: String
Origin()
<<Java Class>>
Category
code.energy.ruler
id: Long
name: String
Category()
<<Java Class>>
Measure
code.energy.ruler
id: Long
date: Date
Measure()
isClean():Boolean
<<Java Class>>
Environment
code.energy.ruler
id: Long
arch: String
name: String
version: String
Environment()
<<Java Class>>
Data
code.energy.ruler
id: Long
time: Long
value: Double
Data()
compareTo(Data):int
equals(Object):boolean
<<Java Class>>
Runtime
code.energy.ruler
id: Long
name: String
version: String
Runtime()
-user
-codes
0..1
0..*
-origin
0..1
-code
0..1
-measures
0..*
-category
0..1
-measure
0..1
-data
0..*
-environment
0..1
-runtime
0..1