SNAIL
TEAM@UNAMUR

Software Normalization Assessment and Improvement Lab

"We research emerging software quality properties through the lenses of software reliability and developers' experience"

https://snail.info.unamur.be/

UNIVERSITÉ DE NAMUR
FACULTÉ D'INFORMATIQUE

NADI
Namur Digital Institute

Prof. Xavier Devroey
(a.k.a. Mr. Testing)

Prof. Benoît Vanderose
(a.k.a. Mr. Quality)

✉ jerome.maquoi@unamur.be

🌐 https://jeromemaquoi.github.io/

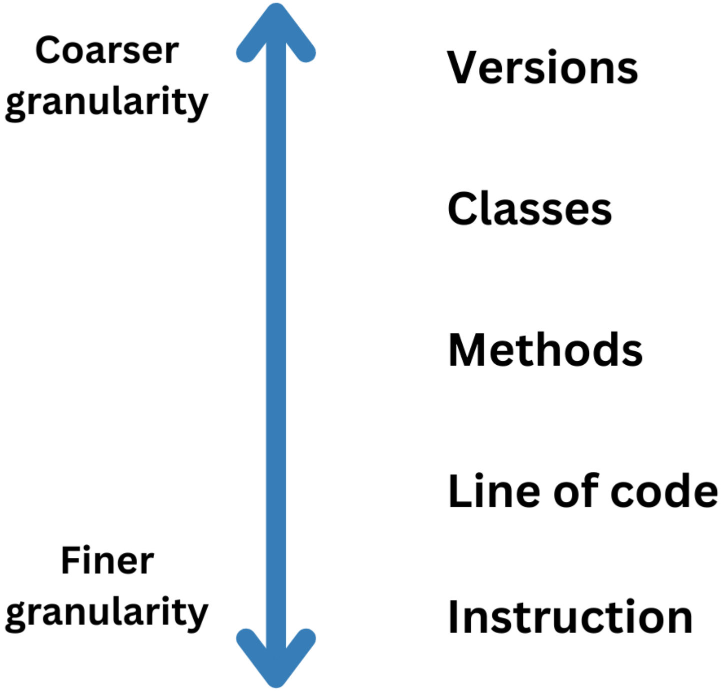
Raising awareness about source code energy consumption

January 23rd, 2026
Journée GT Logiciel Eco-Responsable, Nantes

NADI Namur Digital Institute

UNIVERSITÉ DE NAMUR
FACULTÉ D'INFORMATIQUE

Granularity of analysis



Coarser granularity

Versions

Classes

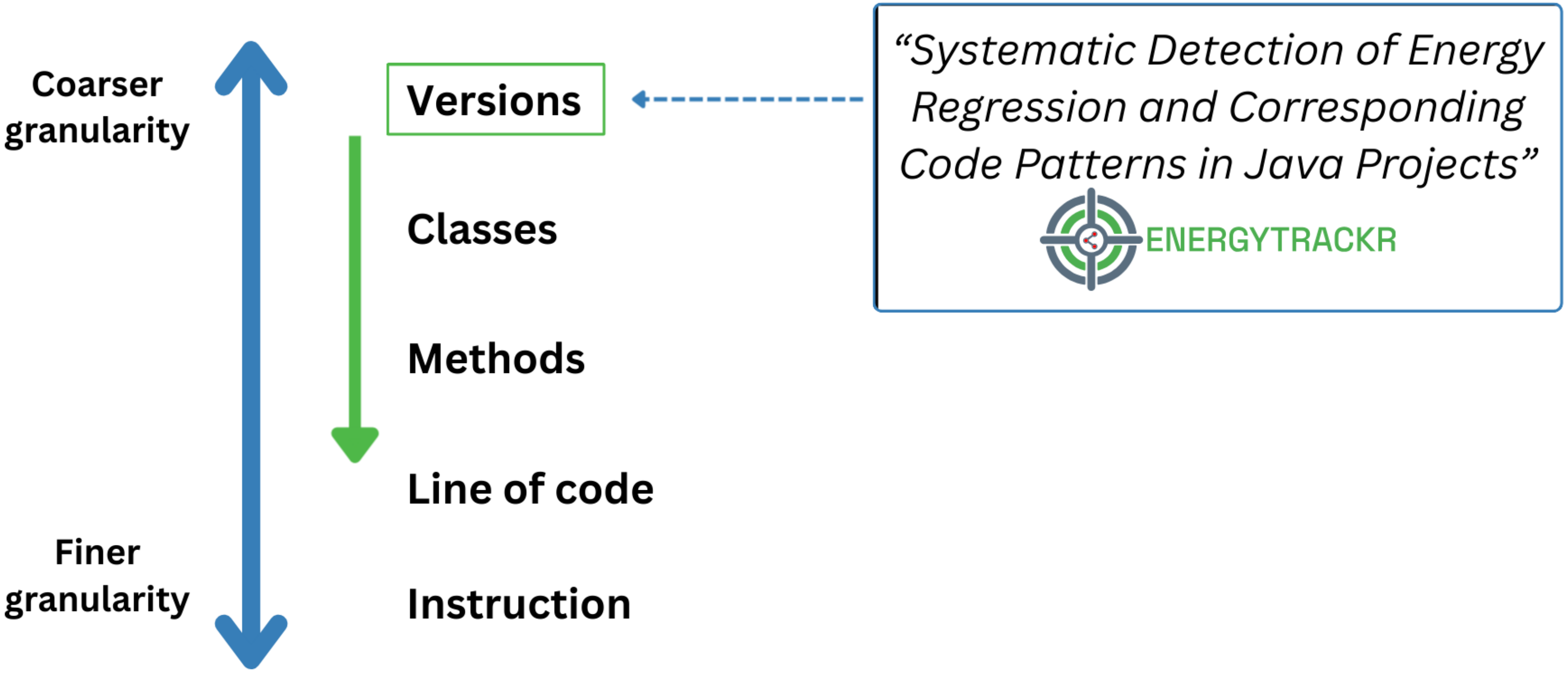
Methods

Line of code

Instruction

Finer granularity

Top-down approach



Coarser granularity

Finer granularity

Versions

Classes

Methods

Line of code

Instruction

“Systematic Detection of Energy Regression and Corresponding Code Patterns in Java Projects”

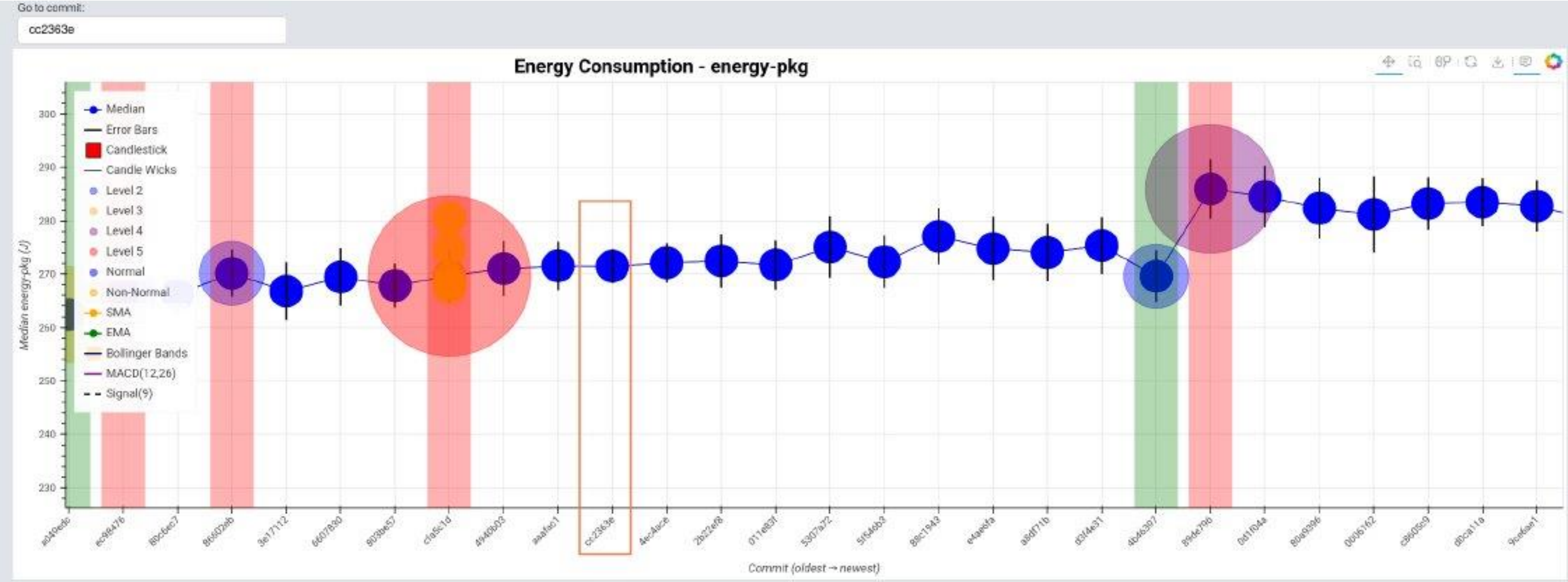
ENERGYTRACKR

EnergyTrackr approach

- Implemented by François Bechet, during his Master Thesis
- Multiple commits energy measurement through test suite execution
- Report with results analysis and flagging potential energy regression commits

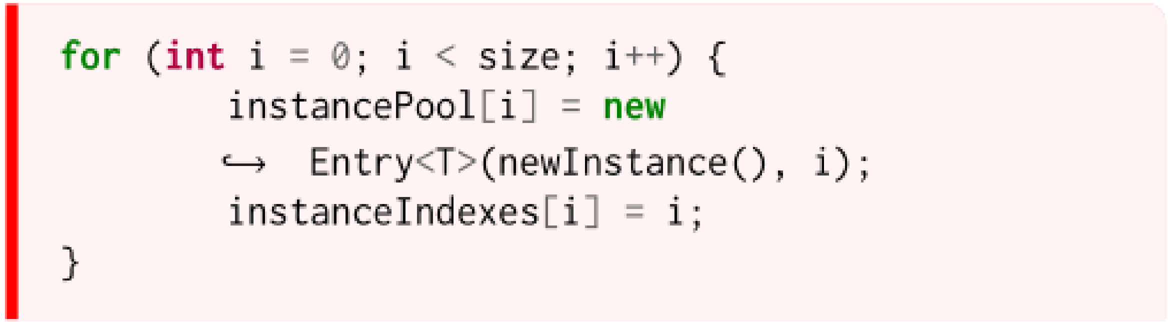
ENERGYTRACKR

Evolution plot example



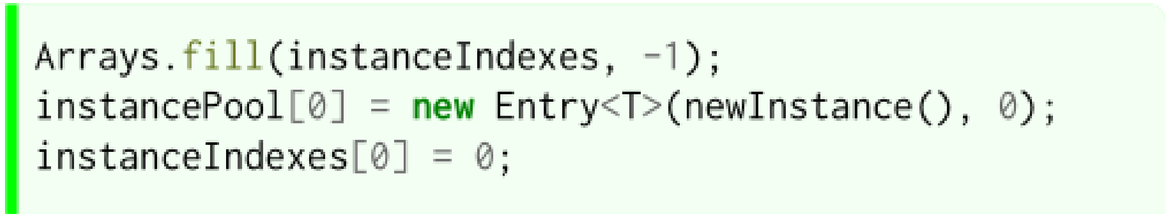
Go to commit: cc2363e

"Eager allocation" pattern example



```
for (int i = 0; i < size; i++) {
        instancePool[i] = new
       ↪  Entry<T>(newInstance(), i);
        instanceIndexes[i] = i;
}
```

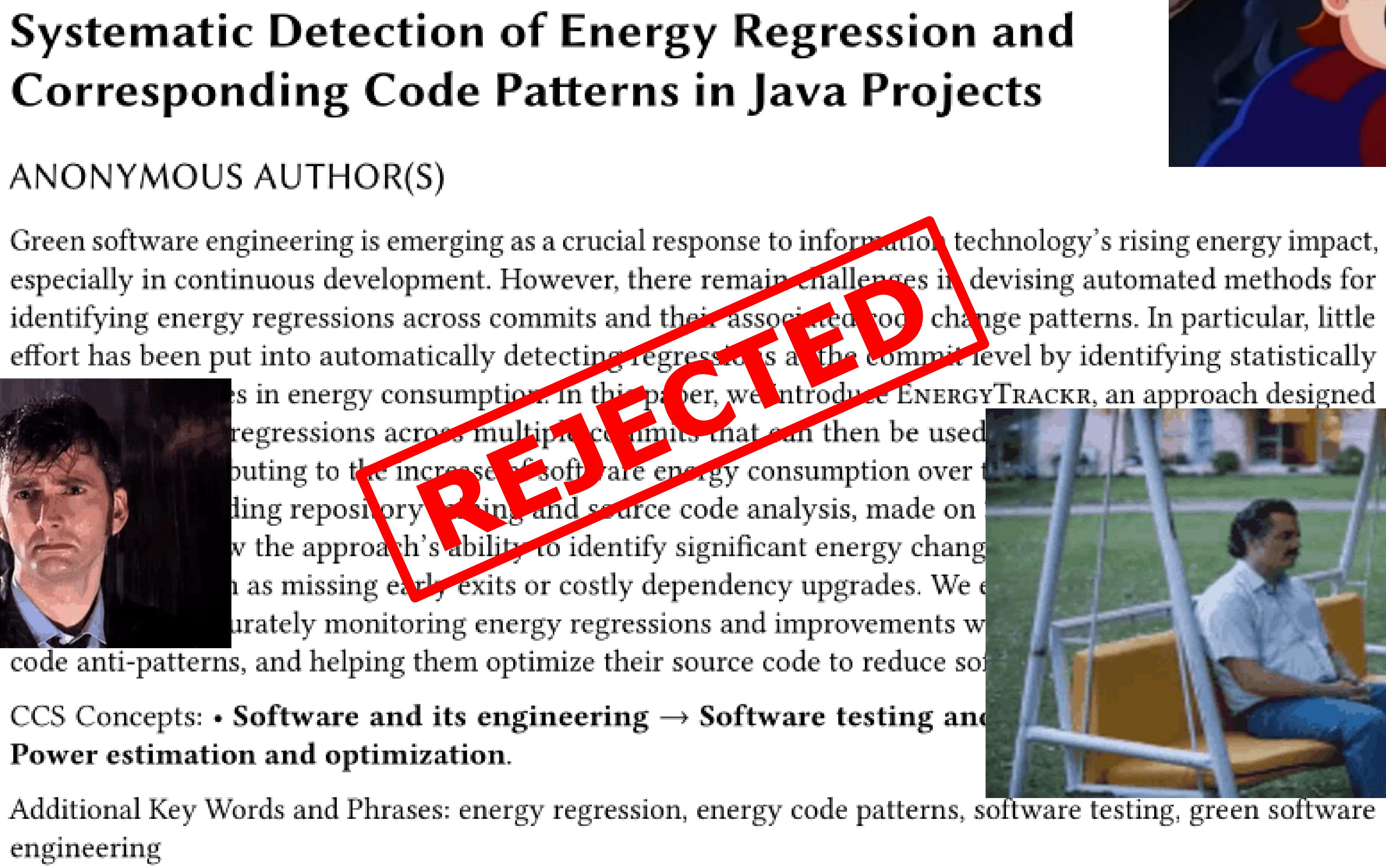
Old code



```
Arrays.fill(instanceIndexes, -1);
instancePool[0] = new Entry<T>(newInstance(), 0);
instanceIndexes[0] = 0;
```

New code

Paper submission at FSE 2026



Systematic Detection of Energy Regression and Corresponding Code Patterns in Java Projects

ANONYMOUS AUTHOR(S)

Green software engineering is emerging as a crucial response to information technology's rising energy impact,
especially in continuous development. However, there remain challenges in devising automated methods for
identifying energy regressions across commits and their associated code change patterns. In particular, little
effort has been put into automatically detecting regressions at the commit level by identifying statistically
es in energy consumption. In this paper, we introduce EnergyTrackr, an approach designed
regressions across multiple commits that can then be used
puting to the increase of software energy consumption over t
ding repository mining and source code analysis, made on
v the approach's ability to identify significant energy chang
n as missing early exits or costly dependency upgrades. We e
urately monitoring energy regressions and improvements w
code anti-patterns, and helping them optimize their source code to reduce so

REJECTED

CCS Concepts: • **Software and its engineering** → **Software testing and**
Power estimation and optimization.

Additional Key Words and Phrases: energy regression, energy code patterns, software testing, green software engineering

Energy Codesumption, Leveraging Test Execution for Source Code Energy Consumption Analysis

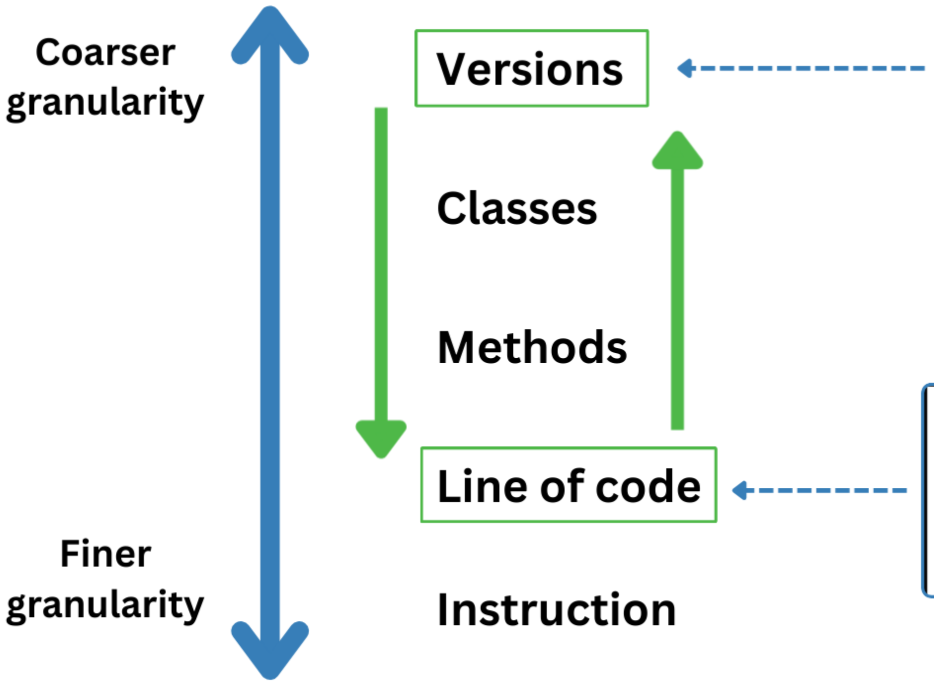
Jérôme MAQUOI
Maxime CAUZ
Benoît VANDEROSE
Xavier DEVROEY

NADI, University of Namur, Belgium

First International Workshop on DevOps for Sustainability, co-located with FSE 2025
Mon 23 - Fri 27 June 2025 Trondheim, Norway

UNIVERSITÉ DE NAMUR

Bottom-up approach



Coarser granularity

Versions

Classes

Methods

Line of code

Instruction

Finer granularity

“Systematic Detection of Energy Regression and Corresponding Code Patterns in Java Projects”

ENERGYTRACKR

“Energy CodeSumption, Leveraging Test Execution for Source Code Energy Consumption Analysis”

RQ: How does the source code of a Java project impact its energy consumption ?

Energy consumption assessment

JoularJX monitors power usage of Java projects at the source code level [2]

Energy consumption measurement

Best practices (Cruz, 2021) :

Zen mode & freeze	Server specs : Ubuntu 22.04.5, 64 Intel(R) Xeon(R) Gold 6326 (2.90GHz) and 256GB of RAM
Warm up the system	5 minutes preliminary test before the measurement
Repeat	30 executions of the projects' test suite
Rest	1 minute cooling down between each test suite execution
Keep it cool	Stable room temperature during the experiment
Automate	Bash script automating : - repositories cloning, - JoularJX agent preparation, - execution of all the steps before, - data storage in a MongoDB

Selected projects

	Spring Boot	Spoon
Version	V3.1.4	V10.4.2
Commit SHA	3ed1f1a064a10e53adc2 ad8c0b46a4b2c148ee21	066f4cf207359e06d309 11a553dedd054aef595c
JDK Version	19	17
Total / Failed / Ignored tests	4217 / 0 / 12	4276 / 0 / 12
Lines of Code (LOC)	23,358	28,739
Class Coverage	76% (795 / 1037)	**97%** (922 / 943)
Method Coverage	70% (4662 / 6630)	**88%** (6691 / 7546)
Line Coverage	68% (16031 / 23,358)	**87%** (25,045 / 28,739)
Branch Coverage	65% (5902 / 9062)	**77%** (10,822 / 14,020)

JoularJX data structure

```
spoon.[...].jdt.JDTBasedSpoonCompilerTest.testOrderCompilationUnits 35
spoon.[...].jdt.JDTBasedSpoonCompiler.buildUnits 418
spoon.[...].jdt.JDTBatchCompiler.getUnits 282
spoon.[...].jdt.TreeBuilderCompiler.buildUnits 82
```

+ 318 J = One **Call Trace**

Data pre-processing

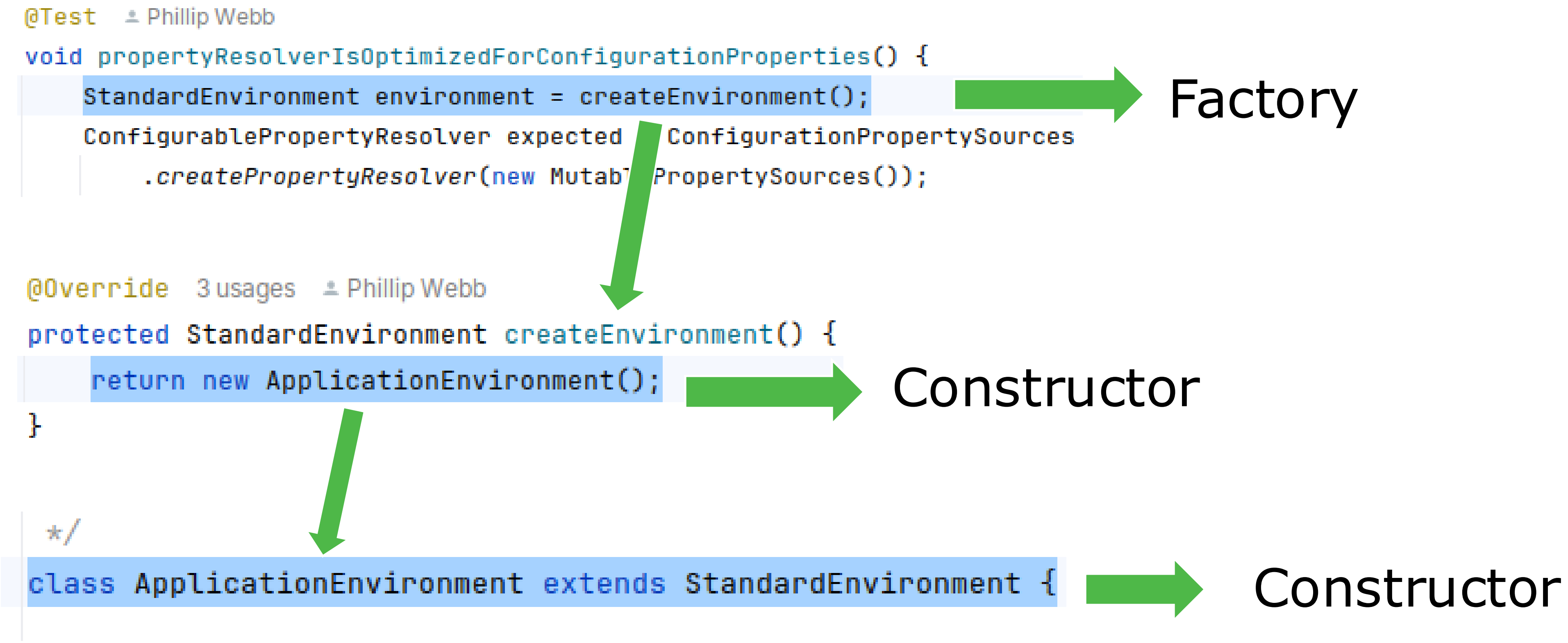
Steps	# remaining CTs	
	Spring Boot	Spoon
Retain only CTs for instance with at least 25 energy data	50	43
Filter outliers with standard deviation	48	40
Shapiro-Wilk test for normality evaluation	27	31

Data analysis

Manual analysis of the 5 most and least energy-intensive CTs

Categorization of each frame's method within the CTs

CT1 example :



```
@Test  Phillip Webb
void propertyResolverIsOptimizedForConfigurationProperties() {
    StandardEnvironment environment = createEnvironment();
    ConfigurablePropertyResolver expected  ConfigurationPropertySources
        .createPropertyResolver(new Mutabl PropertySources());
```

Factory

```
@Override  3 usages  Phillip Webb
protected StandardEnvironment createEnvironment() {
    return new ApplicationEnvironment();
}
```

Constructor

```
 */
class ApplicationEnvironment extends StandardEnvironment {
```

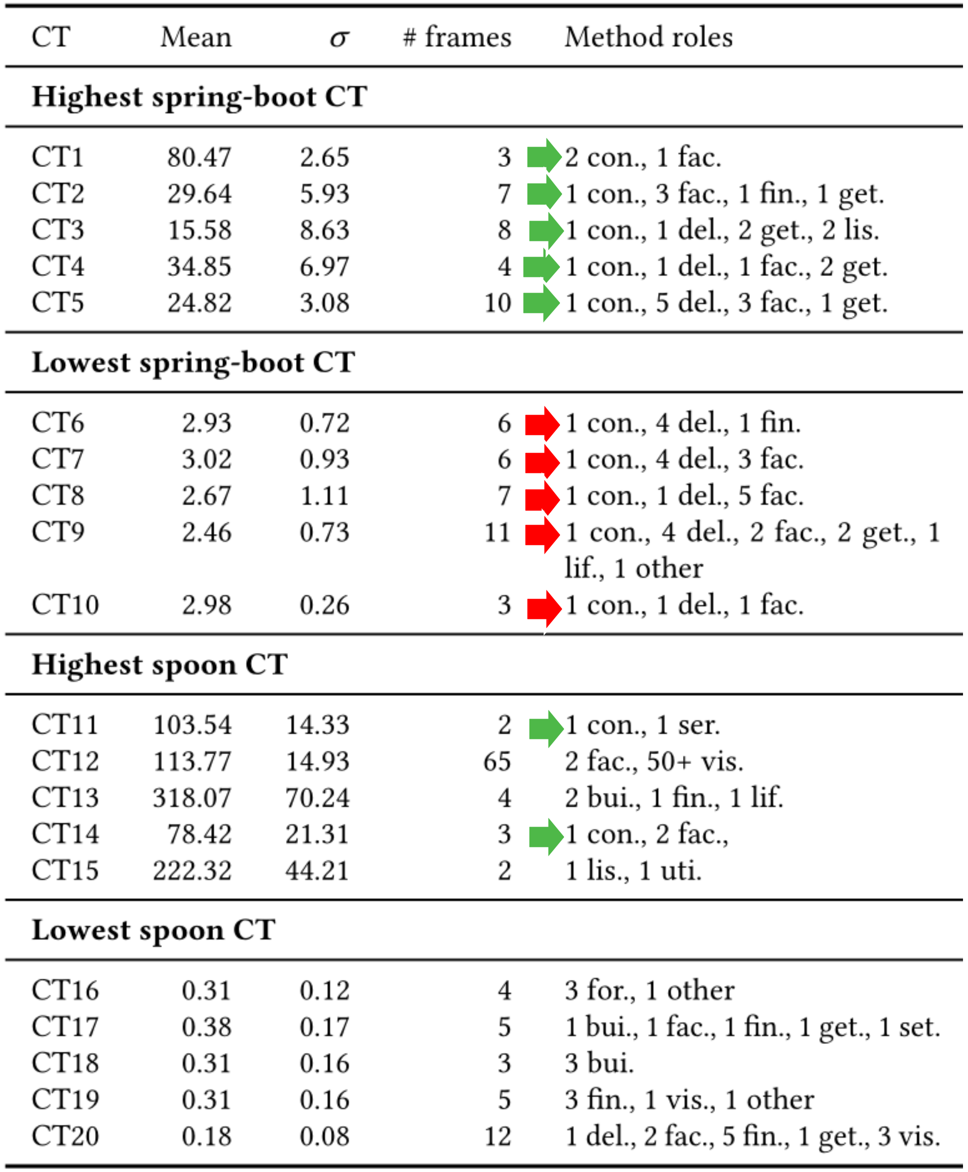
Constructor

Evaluation Results

7 out of 10 most energy-intensive CTs involved constructor

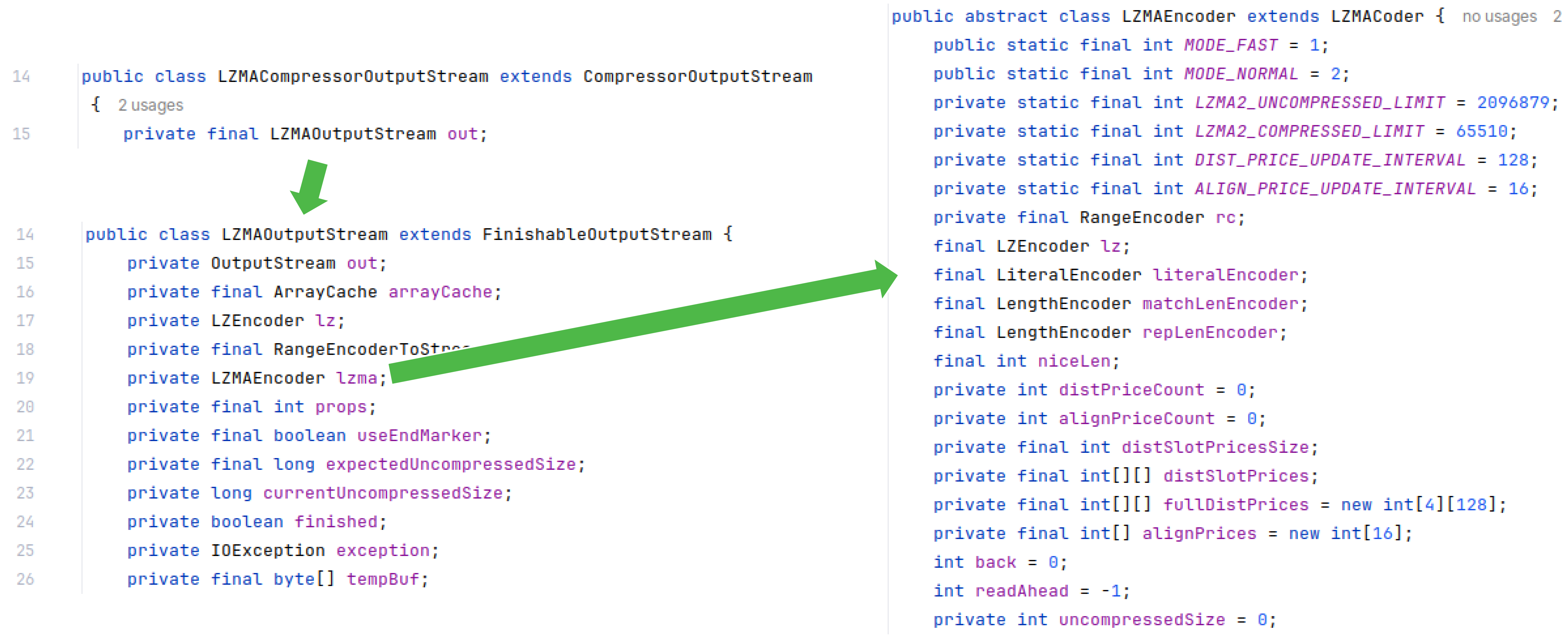
BUT 5 out of 10 least energy-intensive CT's involved constructor

➡ Inspection of the program state and constructor-created attributes



CT	Mean	σ	# frames	Method roles
Highest spring-boot CT				
CT1	80.47	2.65	3	➡ 2 con., 1 fac.
CT2	29.64	5.93	7	➡ 1 con., 3 fac., 1 fin., 1 get.
CT3	15.58	8.63	8	➡ 1 con., 1 del., 2 get., 2 lis.
CT4	34.85	6.97	4	➡ 1 con., 1 del., 1 fac., 2 get.
CT5	24.82	3.08	10	➡ 1 con., 5 del., 3 fac., 1 get.
Lowest spring-boot CT				
CT6	2.93	0.72	6	➡ 1 con., 4 del., 1 fin.
CT7	3.02	0.93	6	➡ 1 con., 4 del., 3 fac.
CT8	2.67	1.11	7	➡ 1 con., 1 del., 5 fac.
CT9	2.46	0.73	11	➡ 1 con., 4 del., 2 fac., 2 get., 1 lif., 1 other
CT10	2.98	0.26	3	➡ 1 con., 1 del., 1 fac.
Highest spoon CT				
CT11	103.54	14.33	2	➡ 1 con., 1 ser.
CT12	113.77	14.93	65	2 fac., 50+ vis.
CT13	318.07	70.24	4	2 bui., 1 fin., 1 lif.
CT14	78.42	21.31	3	➡ 1 con., 2 fac.,
CT15	222.32	44.21	2	1 lis., 1 uti.
Lowest spoon CT				
CT16	0.31	0.12	4	3 for., 1 other
CT17	0.38	0.17	5	1 bui., 1 fac., 1 fin., 1 get., 1 set.
CT18	0.31	0.16	3	3 bui.
CT19	0.31	0.16	5	3 fin., 1 vis., 1 other
CT20	0.18	0.08	12	1 del., 2 fac., 5 fin., 1 get., 3 vis.

Example of Highest CT with CT11

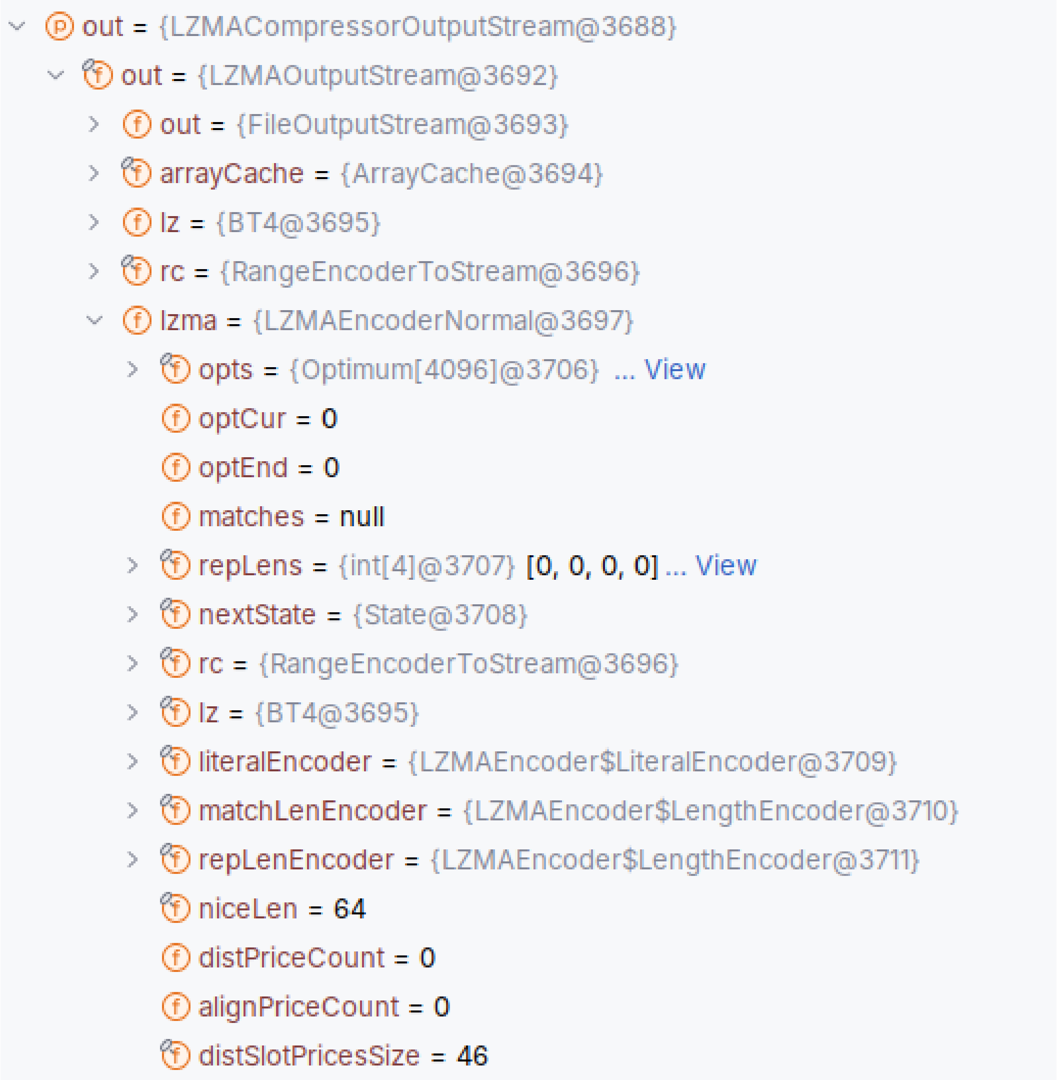


```java
public class LZMACompressorOutputStream extends CompressorOutputStream
 {  2 usages
    private final LZMAOutputStream out;
```

```java
public class LZMAOutputStream extends FinishableOutputStream {
    private OutputStream out;
    private final ArrayCache arrayCache;
    private LZEncoder lz;
    private final RangeEncoderToStr
    private LZMAEncoder lzma;
    private final int props;
    private final boolean useEndMarker;
    private final long expectedUncompressedSize;
    private long currentUncompressedSize;
    private boolean finished;
    private IOException exception;
    private final byte[] tempBuf;
```

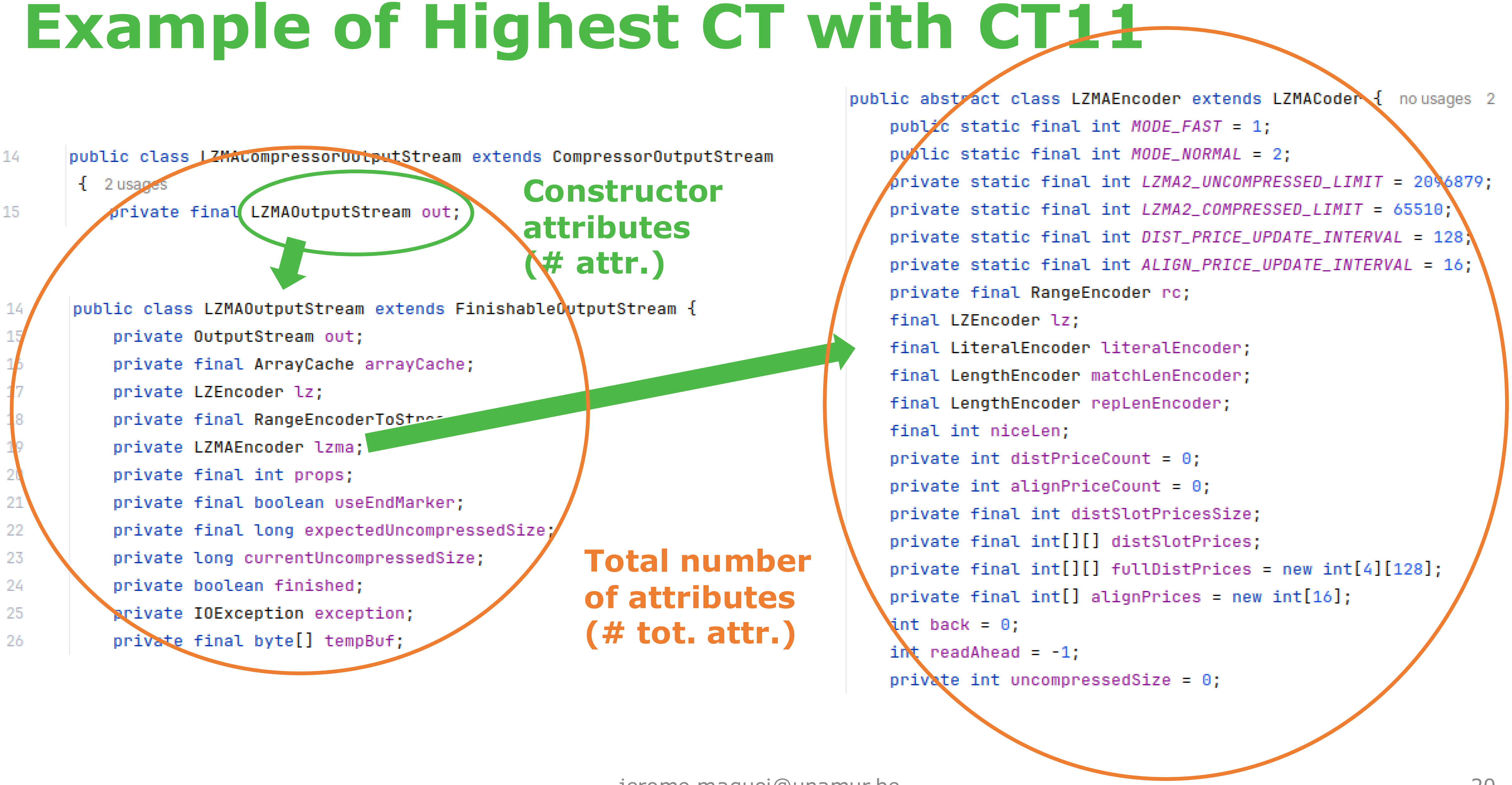
```java
public abstract class LZMAEncoder extends LZMACoder {  no usages  2
    public static final int MODE_FAST = 1;
    public static final int MODE_NORMAL = 2;
    private static final int LZMA2_UNCOMPRESSED_LIMIT = 2096879;
    private static final int LZMA2_COMPRESSED_LIMIT = 65510;
    private static final int DIST_PRICE_UPDATE_INTERVAL = 128;
    private static final int ALIGN_PRICE_UPDATE_INTERVAL = 16;
    private final RangeEncoder rc;
    final LZEncoder lz;
    final LiteralEncoder literalEncoder;
    final LengthEncoder matchLenEncoder;
    final LengthEncoder repLenEncoder;
    final int niceLen;
    private int distPriceCount = 0;
    private int alignPriceCount = 0;
    private final int distSlotPricesSize;
    private final int[][] distSlotPrices;
    private final int[][] fullDistPrices = new int[4][128];
    private final int[] alignPrices = new int[16];
    int back = 0;
    int readAhead = -1;
    private int uncompressedSize = 0;
```

Example of Highest CT with CT11



```
⌄ ⓟ out = {LZMACompressorOutputStream@3688}
    ⌄ ⓕ out = {LZMAOutputStream@3692}
        › ⓕ out = {FileOutputStream@3693}
        › ⓕ arrayCache = {ArrayCache@3694}
        › ⓕ lz = {BT4@3695}
        › ⓕ rc = {RangeEncoderToStream@3696}
        ⌄ ⓕ lzma = {LZMAEncoderNormal@3697}
            › ⓕ opts = {Optimum[4096]@3706} ... View
              ⓕ optCur = 0
              ⓕ optEnd = 0
              ⓕ matches = null
            › ⓕ repLens = {int[4]@3707} [0, 0, 0, 0] ... View
            › ⓕ nextState = {State@3708}
            › ⓕ rc = {RangeEncoderToStream@3696}
            › ⓕ lz = {BT4@3695}
            › ⓕ literalEncoder = {LZMAEncoder$LiteralEncoder@3709}
            › ⓕ matchLenEncoder = {LZMAEncoder$LengthEncoder@3710}
            › ⓕ repLenEncoder = {LZMAEncoder$LengthEncoder@3711}
              ⓕ niceLen = 64
              ⓕ distPriceCount = 0
              ⓕ alignPriceCount = 0
              ⓕ distSlotPricesSize = 46
```

Example of Highest CT with CT11



```
public class LZMACompressorOutputStream extends CompressorOutputStream
 {  2 usages
    private final LZMAOutputStream out;
```

Constructor attributes (# attr.)

```
public class LZMAOutputStream extends FinishableOutputStream {
    private OutputStream out;
    private final ArrayCache arrayCache;
    private LZEncoder lz;
    private final RangeEncoderToStr
    private LZMAEncoder lzma;
    private final int props;
    private final boolean useEndMarker;
    private final long expectedUncompressedSize;
    private long currentUncompressedSize;
    private boolean finished;
    private IOException exception;
    private final byte[] tempBuf;
```

Total number of attributes (# tot. attr.)

```
public abstract class LZMAEncoder extends LZMACoder {  no usages  2
    public static final int MODE_FAST = 1;
    public static final int MODE_NORMAL = 2;
    private static final int LZMA2_UNCOMPRESSED_LIMIT = 2096879;
    private static final int LZMA2_COMPRESSED_LIMIT = 65510;
    private static final int DIST_PRICE_UPDATE_INTERVAL = 128;
    private static final int ALIGN_PRICE_UPDATE_INTERVAL = 16;
    private final RangeEncoder rc;
    final LZEncoder lz;
    final LiteralEncoder literalEncoder;
    final LengthEncoder matchLenEncoder;
    final LengthEncoder repLenEncoder;
    final int niceLen;
    private int distPriceCount = 0;
    private int alignPriceCount = 0;
    private final int distSlotPricesSize;
    private final int[][] distSlotPrices;
    private final int[][] fullDistPrices = new int[4][128];
    private final int[] alignPrices = new int[16];
    int back = 0;
    int readAhead = -1;
    private int uncompressedSize = 0;
```

Evaluation Results

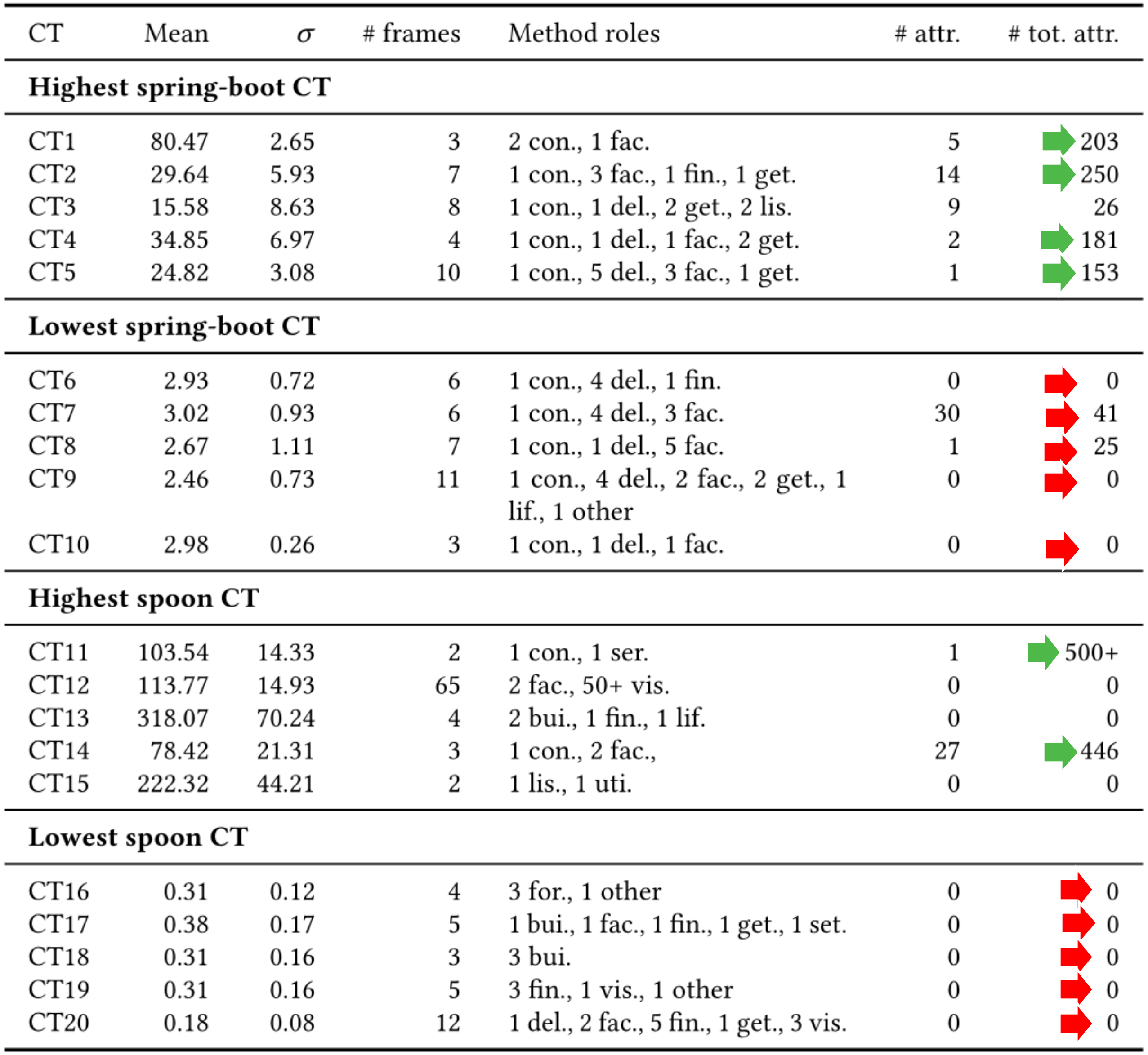
6 out of 7 most energy-consuming constructors produced between 153 and 500 attributes

Least energy-intensive ones generated only 0 to 41 attributes

Spearman's test (ρ = 0.439, $p-value$ = 0.052) : moderate correlation not statistically significant

Kendall's test (τ = 0.4308, $p-value$ = 0.0138) : moderate correlation statistically significant

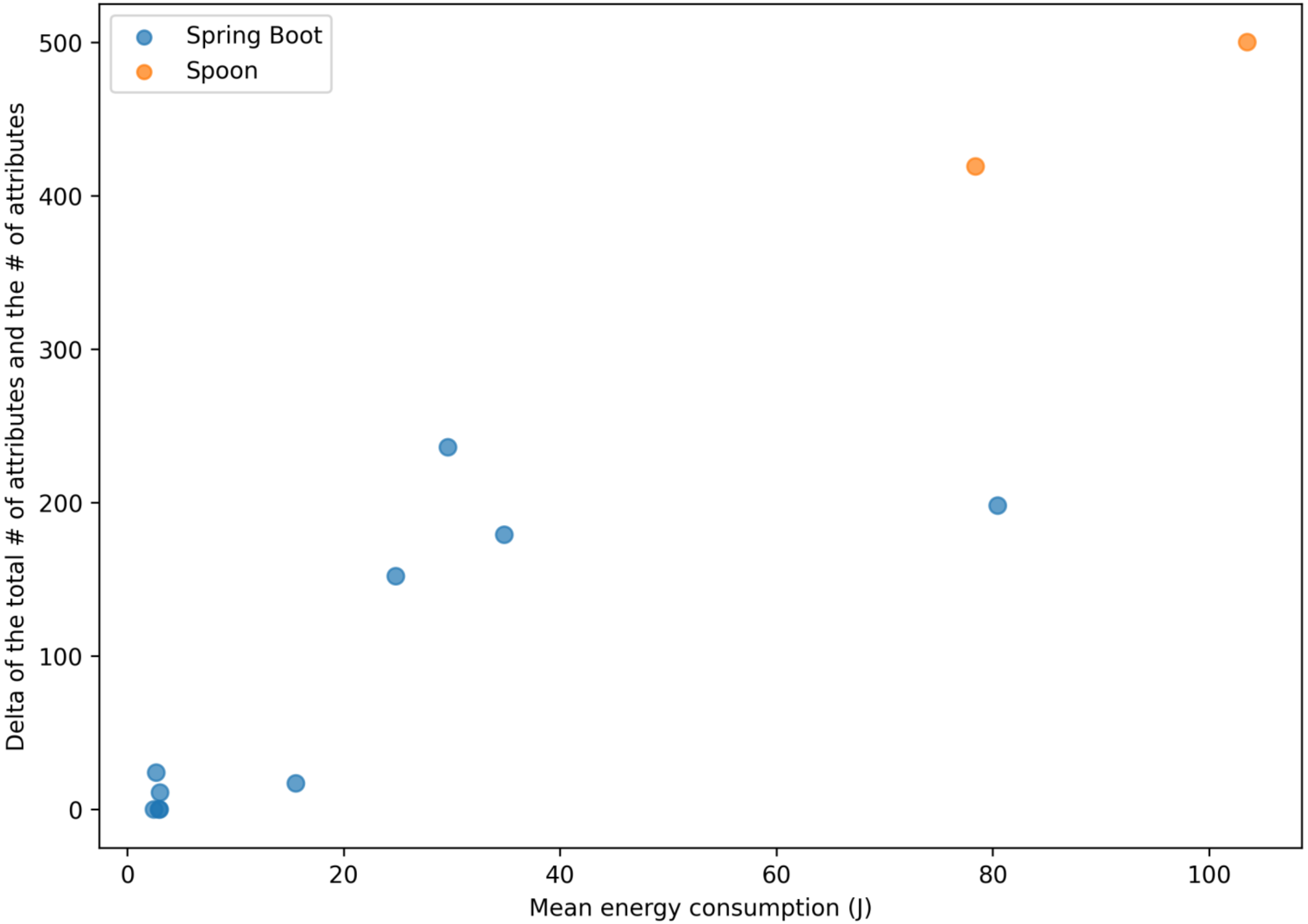
Suggestion of hidden complexity in constructors' attributes among the highest CTs



CT	Mean	σ	# frames	Method roles	# attr.	# tot. attr.
Highest spring-boot CT						
CT1	80.47	2.65	3	2 con., 1 fac.	5	➡ 203
CT2	29.64	5.93	7	1 con., 3 fac., 1 fin., 1 get.	14	➡ 250
CT3	15.58	8.63	8	1 con., 1 del., 2 get., 2 lis.	9	26
CT4	34.85	6.97	4	1 con., 1 del., 1 fac., 2 get.	2	➡ 181
CT5	24.82	3.08	10	1 con., 5 del., 3 fac., 1 get.	1	➡ 153
Lowest spring-boot CT						
CT6	2.93	0.72	6	1 con., 4 del., 1 fin.	0	➡ 0
CT7	3.02	0.93	6	1 con., 4 del., 3 fac.	30	➡ 41
CT8	2.67	1.11	7	1 con., 1 del., 5 fac.	1	➡ 25
CT9	2.46	0.73	11	1 con., 4 del., 2 fac., 2 get., 1 lif., 1 other	0	➡ 0
CT10	2.98	0.26	3	1 con., 1 del., 1 fac.	0	➡ 0
Highest spoon CT						
CT11	103.54	14.33	2	1 con., 1 ser.	1	➡ 500+
CT12	113.77	14.93	65	2 fac., 50+ vis.	0	0
CT13	318.07	70.24	4	2 bui., 1 fin., 1 lif.	0	0
CT14	78.42	21.31	3	1 con., 2 fac.,	27	➡ 446
CT15	222.32	44.21	2	1 lis., 1 uti.	0	0
Lowest spoon CT						
CT16	0.31	0.12	4	3 for., 1 other	0	➡ 0
CT17	0.38	0.17	5	1 bui., 1 fac., 1 fin., 1 get., 1 set.	0	➡ 0
CT18	0.31	0.16	3	3 bui.	0	➡ 0
CT19	0.31	0.16	5	3 fin., 1 vis., 1 other	0	➡ 0
CT20	0.18	0.08	12	1 del., 2 fac., 5 fin., 1 get., 3 vis.	0	➡ 0

Evaluation Results

Relationship between mean energy consumption and the difference between # tot. attr. and # attr.



⚡ 🔗

Total number of attributes

−

Number of constructor attributes

Energy costs may be caused by the quantity and complexity of generated attributes inside constructors

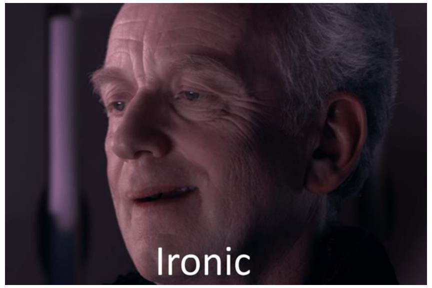
Future Work

- Objective and systematic method categorization
- Automatic identification and counting of attributes ⌛
- Expansion of the analysis to other projects ⌛
- Integration of static analysis

Energy Codesumption extension

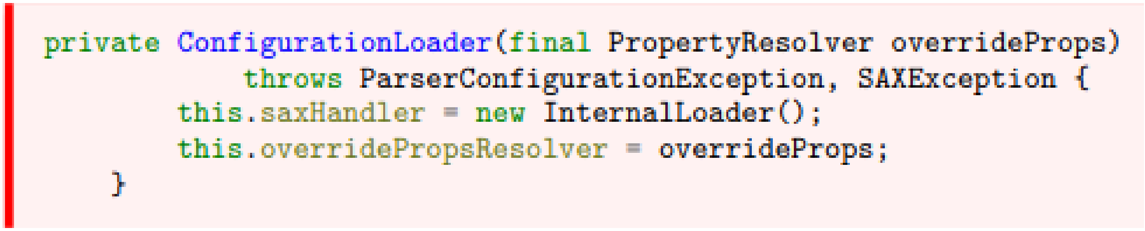
Automatic identification and counting of attributes

- Source code instrumentation with Spoon

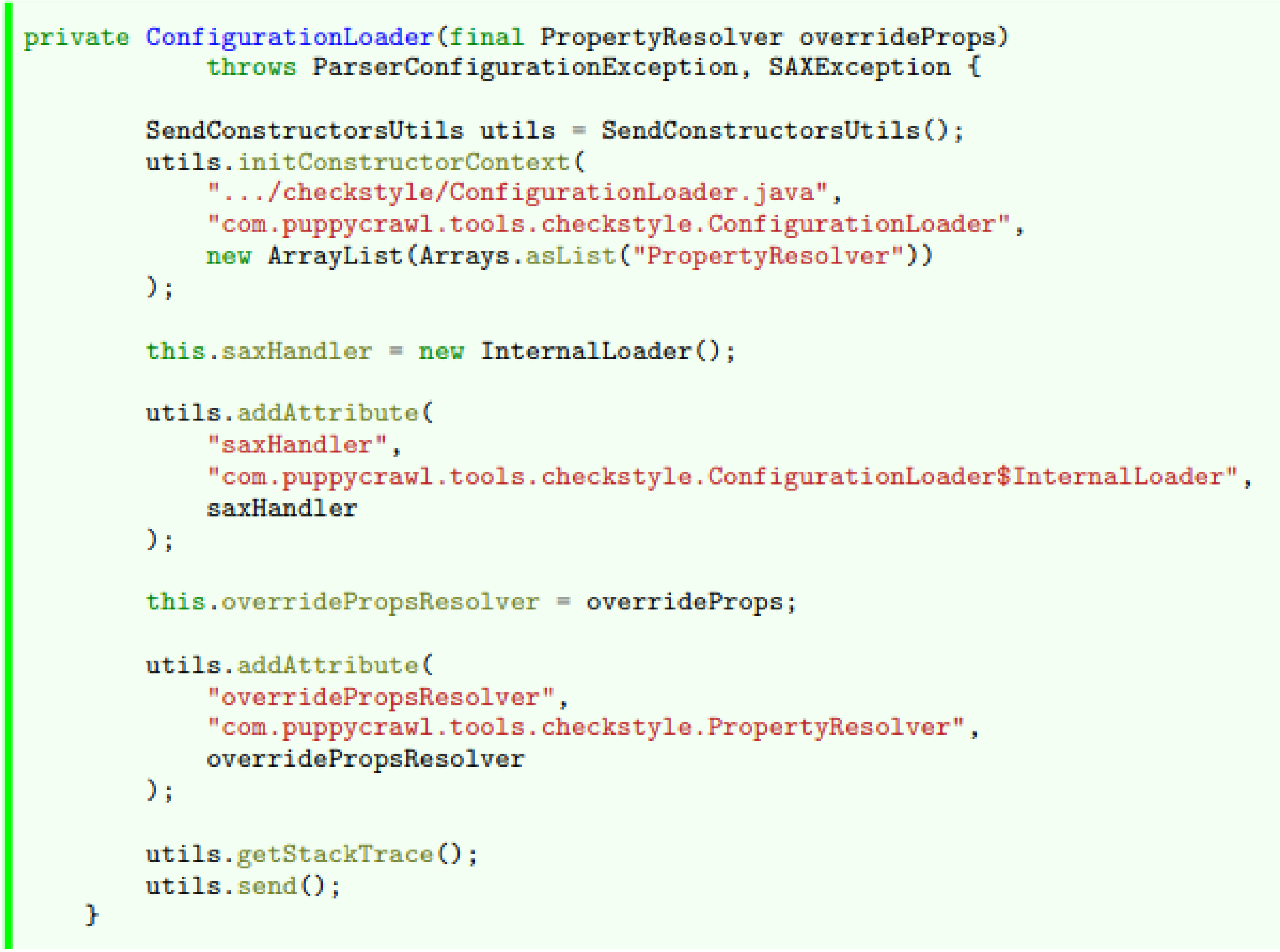


- Objective : automatically gather constructor initialization data to count the number of initialized attributes for each constructor call

Example of Spoon instrumentation

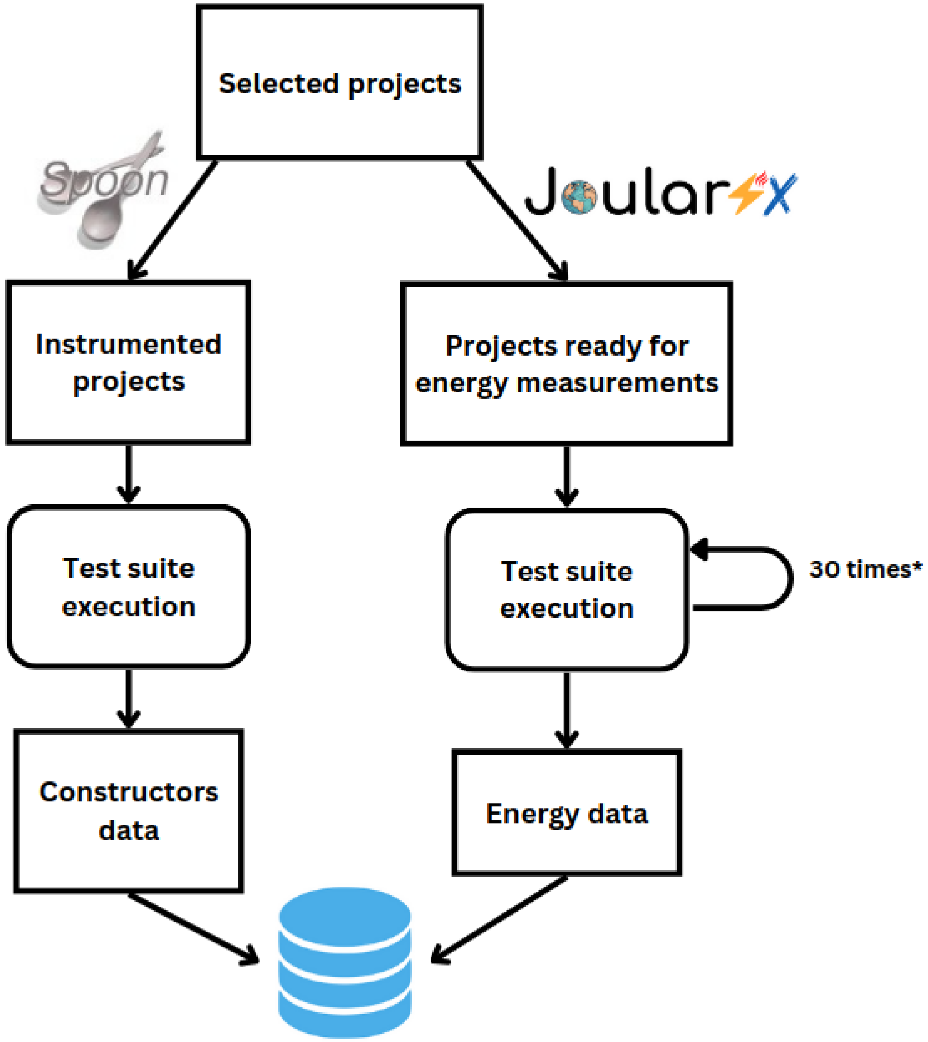


```
private ConfigurationLoader(final PropertyResolver overrideProps)
            throws ParserConfigurationException, SAXException {
        this.saxHandler = new InternalLoader();
        this.overridePropsResolver = overrideProps;
    }
```



```
private ConfigurationLoader(final PropertyResolver overrideProps)
            throws ParserConfigurationException, SAXException {

        SendConstructorsUtils utils = SendConstructorsUtils();
        utils.initConstructorContext(
            ".../checkstyle/ConfigurationLoader.java",
            "com.puppycrawl.tools.checkstyle.ConfigurationLoader",
            new ArrayList(Arrays.asList("PropertyResolver"))
        );

        this.saxHandler = new InternalLoader();

        utils.addAttribute(
            "saxHandler",
            "com.puppycrawl.tools.checkstyle.ConfigurationLoader$InternalLoader",
            saxHandler
        );

        this.overridePropsResolver = overrideProps;

        utils.addAttribute(
            "overridePropsResolver",
            "com.puppycrawl.tools.checkstyle.PropertyResolver",
            overridePropsResolver
        );

        utils.getStackTrace();
        utils.send();
    }
```

Methodology



Selected projects

Spoon

JoularJX

Instrumented projects

Projects ready for energy measurements

Test suite execution

Test suite execution

30 times*

Constructors data

Energy data

Ultimate goal

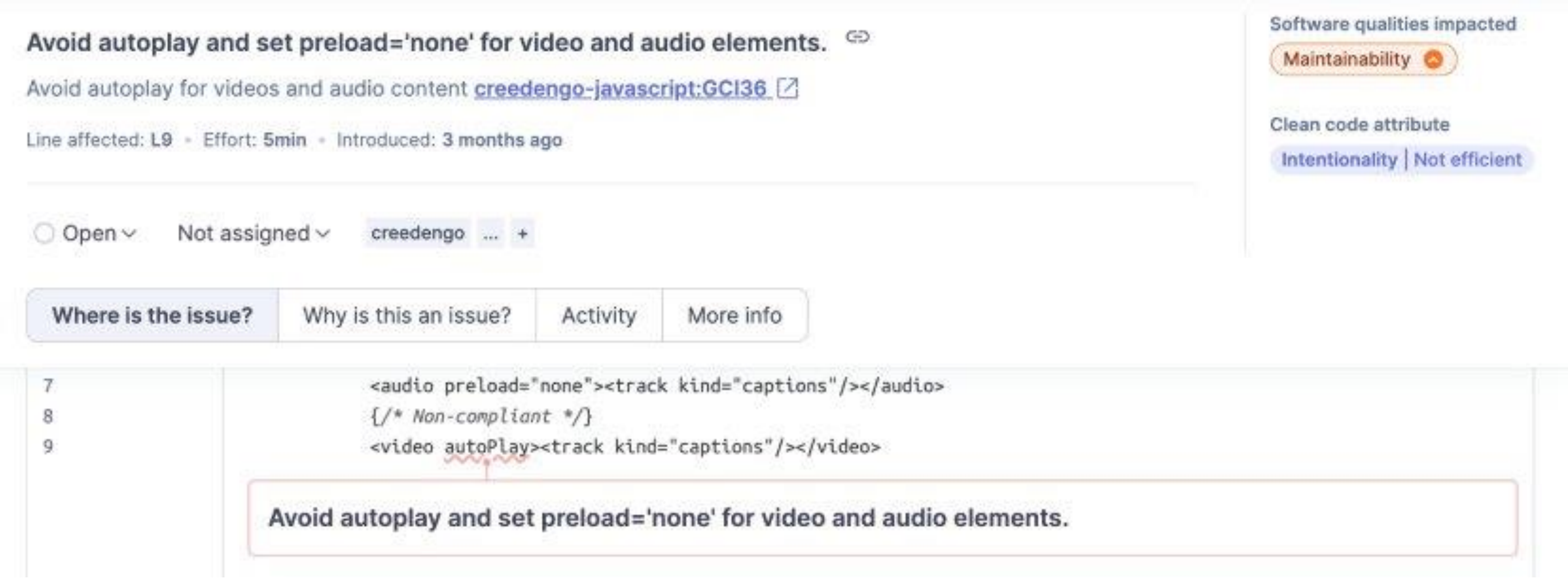
- If possible, define a good practice in terms of constructor usage

- Implement it into the Creedengo project

#CREEDENGO

A Green Code Initiative project

https://green-code-initiative.org/



Avoid autoplay and set preload='none' for video and audio elements.

Avoid autoplay for videos and audio content creedengo-javascript:GCI36

Line affected: L9 · Effort: 5min · Introduced: 3 months ago

Software qualities impacted

Maintainability

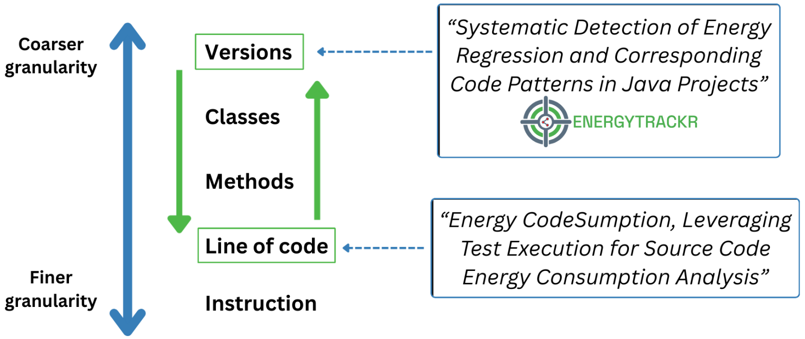
Clean code attribute

Intentionality | Not efficient

```
7                 <audio preload="none"><track kind="captions"/></audio>
8                 {/* Non-compliant */}
9                 <video autoPlay><track kind="captions"/></video>
```

Avoid autoplay and set preload='none' for video and audio elements.

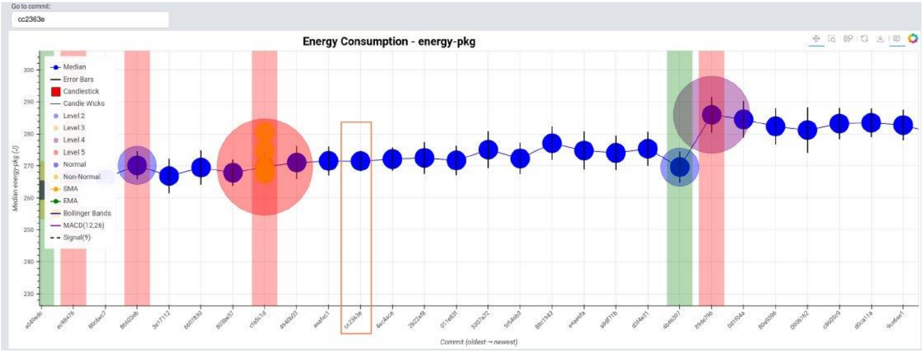
Wrap up

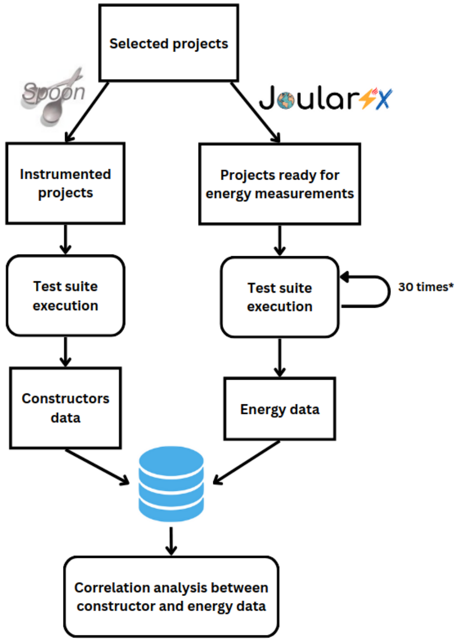


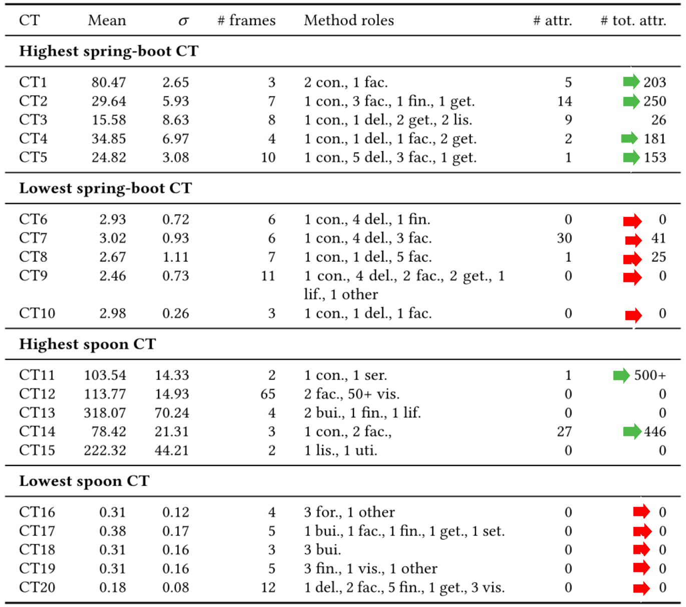
"Systematic Detection of Energy Regression and Corresponding Code Patterns in Java Projects"

"Energy CodeSumption, Leveraging Test Execution for Source Code Energy Consumption Analysis"

ENERGYTRACKR







CT	Mean	σ	# frames	Method roles	# attr.	# tot. attr.
Highest spring-boot CT						
CT1	80.47	2.65	3	2 con., 1 fac.	5	➡ 203
CT2	29.64	5.93	7	1 con., 3 fac., 1 fin., 1 get.	14	➡ 250
CT3	15.58	8.63	8	1 con., 1 del., 2 get., 2 lis.	9	26
CT4	34.85	6.97	4	1 con., 1 del., 1 fac., 2 get.	2	➡ 181
CT5	24.82	3.08	10	1 con., 5 del., 3 fac., 1 get.	1	➡ 153
Lowest spring-boot CT						
CT6	2.93	0.72	6	1 con., 4 del., 1 fin.	0	➡ 0
CT7	3.02	0.93	6	1 con., 4 del., 3 fac.	30	➡ 41
CT8	2.67	1.11	7	1 con., 1 del., 5 fac.	1	➡ 25
CT9	2.46	0.73	11	1 con., 4 del., 2 fac., 2 get., 1 lif., 1 other	0	➡ 0
CT10	2.98	0.26	3	1 con., 1 del., 1 fac.	0	➡ 0
Highest spoon CT						
CT11	103.54	14.33	2	1 con., 1 ser.	1	➡ 500+
CT12	113.77	14.93	65	2 fac., 50+ vis.	0	0
CT13	318.07	70.24	4	2 bui., 1 fin., 1 lif.	0	0
CT14	78.42	21.31	3	1 con., 2 fac.,	27	➡ 446
CT15	222.32	44.21	2	1 lis., 1 uti.	0	0
Lowest spoon CT						
CT16	0.31	0.12	4	3 for., 1 other	0	➡ 0
CT17	0.38	0.17	5	1 bui., 1 fac., 1 fin., 1 get., 1 set.	0	➡ 0
CT18	0.31	0.16	3	3 bui.	0	➡ 0
CT19	0.31	0.16	5	3 fin., 1 vis., 1 other	0	➡ 0
CT20	0.18	0.08	12	1 del., 2 fac., 5 fin., 1 get., 3 vis.	0	➡ 0