

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



Introduction

Developers can recognize software energy consumption challenges [1]

BUT lack of knowledge on how to reduce this consumption



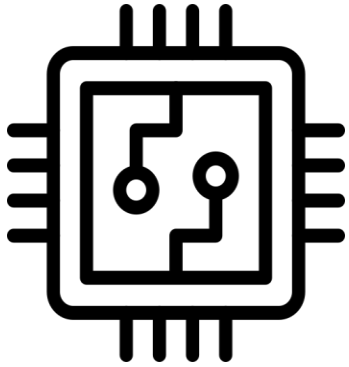
Need for energy
measurement methods



Need for problematic code
constructs identification

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

Energy consumption assessment



Joular⚡✂

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

Evaluation Setup

Energy consumption measurement

Best practices [4] :

Zen mode & freeze

Warm up the system

Repeat

Rest

Keep it cool

Automate

Server specs : Ubuntu 22.04.5, 64 Intel(R) Xeon(R) Gold 6326 (2.90GHz) and 256GB of RAM

5 minutes preliminary test before the measurement

30 executions of the projects' test suite

1 minute cooling down between each test suite execution

Stable room temperature during the experiment

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

Collected JoularJX data:

Stacktraces + energy consumption = **Call Traces** (CTs)

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

+ 318 J **=** One CT

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();
```

➡ Builder

```
    ConfigurablePropertyResolver expected = ConfigurationPropertySources  
        .createPropertyResolver(new MutablePropertySources());
```

```
@Override 3 usages ⚡ Phillip Webb
```

```
protected StandardEnvironment createEnvironment() {
```

```
    return new ApplicationEnvironment();
```

➡ Constructor

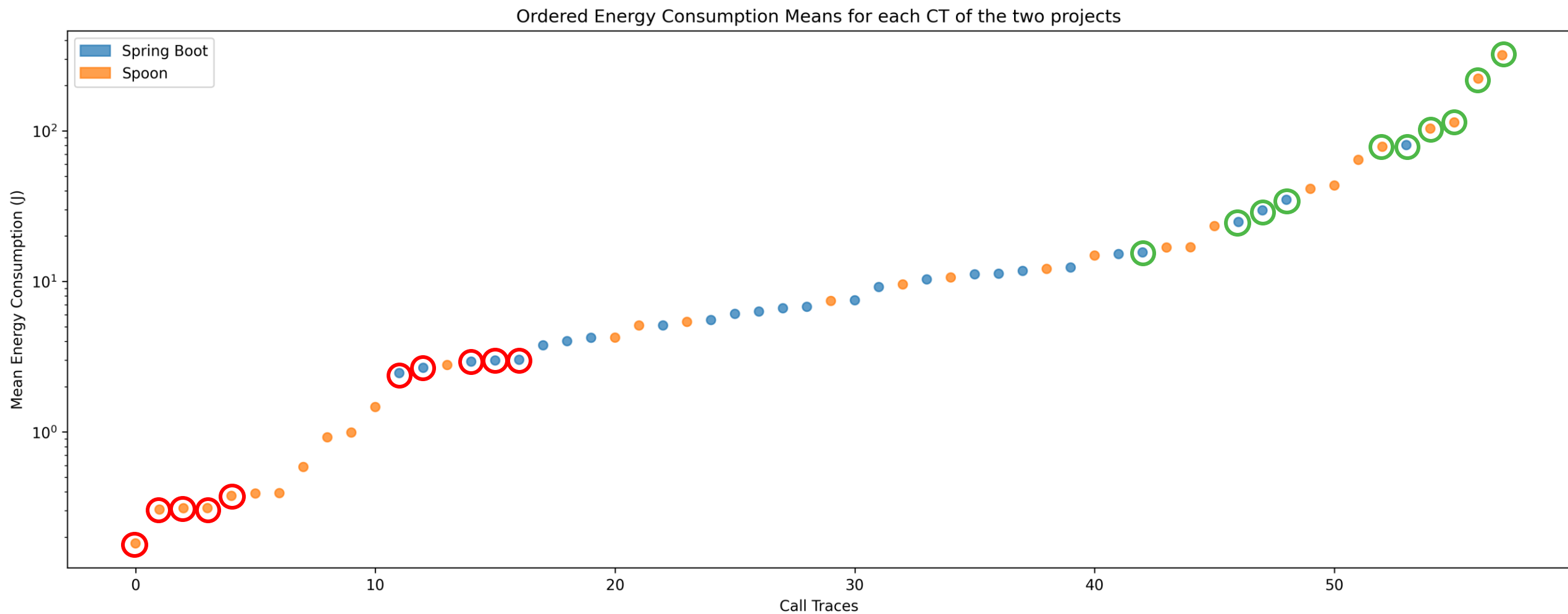
```
}
```

```
*/
```

```
class ApplicationEnvironment extends StandardEnvironment {
```

➡ Constructor


Evaluation Results



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

```
14 public class LZMACompressorOutputStream extends CompressorOutputStream
15 { 2 usages
    private final LZMAOutputStream out;
```



```
14 public class LZMAOutputStream extends FinishableOutputStream {
15     private OutputStream out;
16     private final ArrayCache arrayCache;
17     private LZEncoder lz;
18     private final RangeEncoderToString;
19     private LZMAEncoder lzma;
20     private final int props;
21     private final boolean useEndMarker;
22     private final long expectedUncompressedSize;
23     private long currentUncompressedSize;
24     private boolean finished;
25     private IOException exception;
26     private final byte[] tempBuf;
```

```
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

```

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

```

Example of Highest CT with CT11

```
14 public class LZMACompressorOutputStream extends CompressorOutputStream
```

```
15 { 2 usages  
    private final LZMAOutputStream out;
```

**Constructor
attributes
(# attr.)**

```
14 public class LZMAOutputStream extends FinishableOutputStream {
```

```
15     private OutputStream out;  
16     private final ArrayCache arrayCache;  
17     private LZEncoder lz;  
18     private final RangeEncoderToString  
19     private LZMAEncoder lzma;  
20     private final int props;  
21     private final boolean useEndMarker;  
22     private final long expectedUncompressedSize;  
23     private long currentUncompressedSize;  
24     private boolean finished;  
25     private IOException exception;  
26     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 ($\rho = 0.439$, $p\text{-value} = 0.052$) : moderate correlation not statistically significant

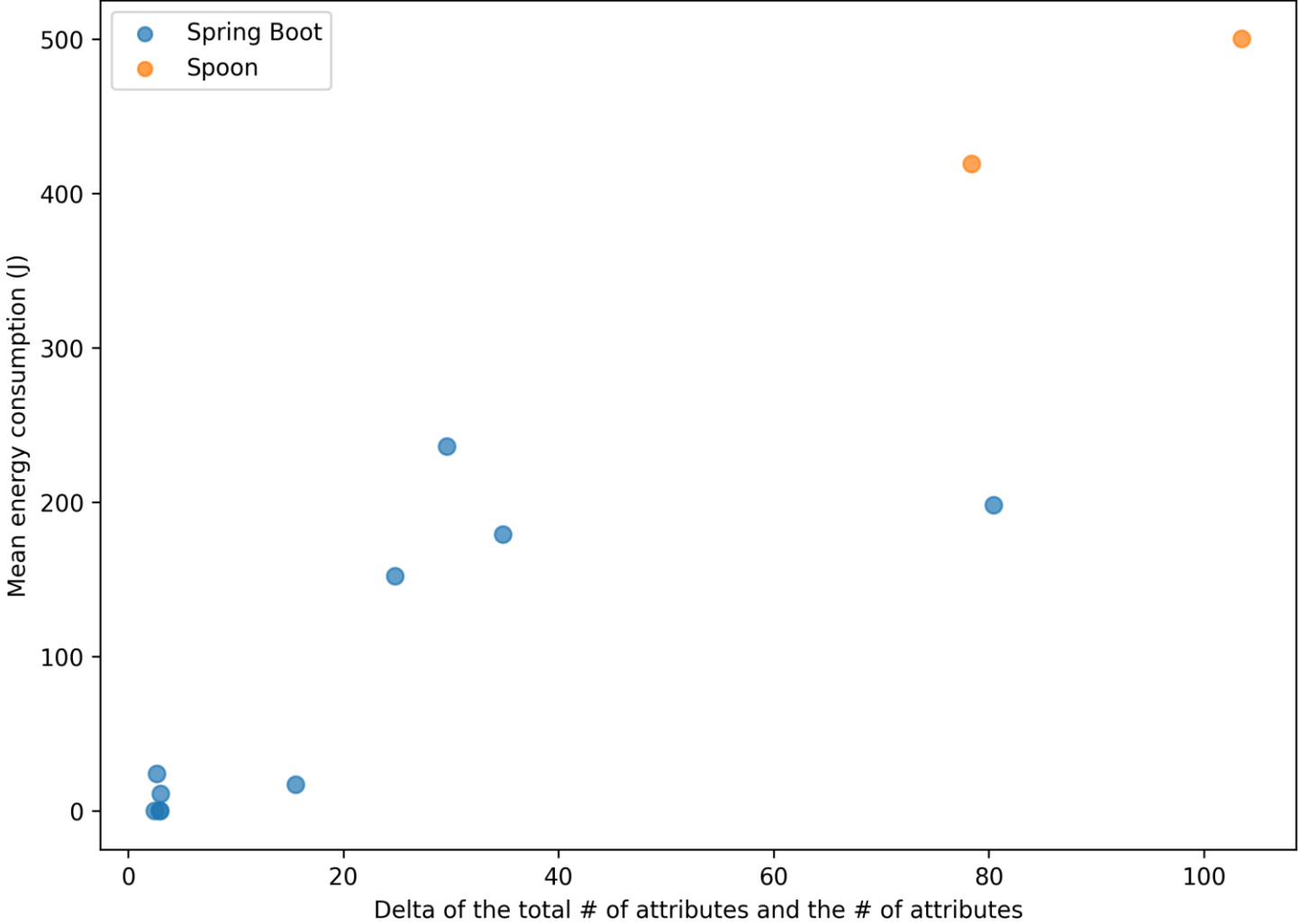
Kendall's test ($\tau = 0.4308$, $p\text{-value} = 0.0138$) : moderate correlation statistically significant

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.



$$\begin{array}{c} \text{Total number of attributes} \\ - \\ \text{Number of constructor} \\ \text{attributes} \end{array}$$



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
- Analysis of multiple commits



Software Normalization Assessment and Improvement Lab

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

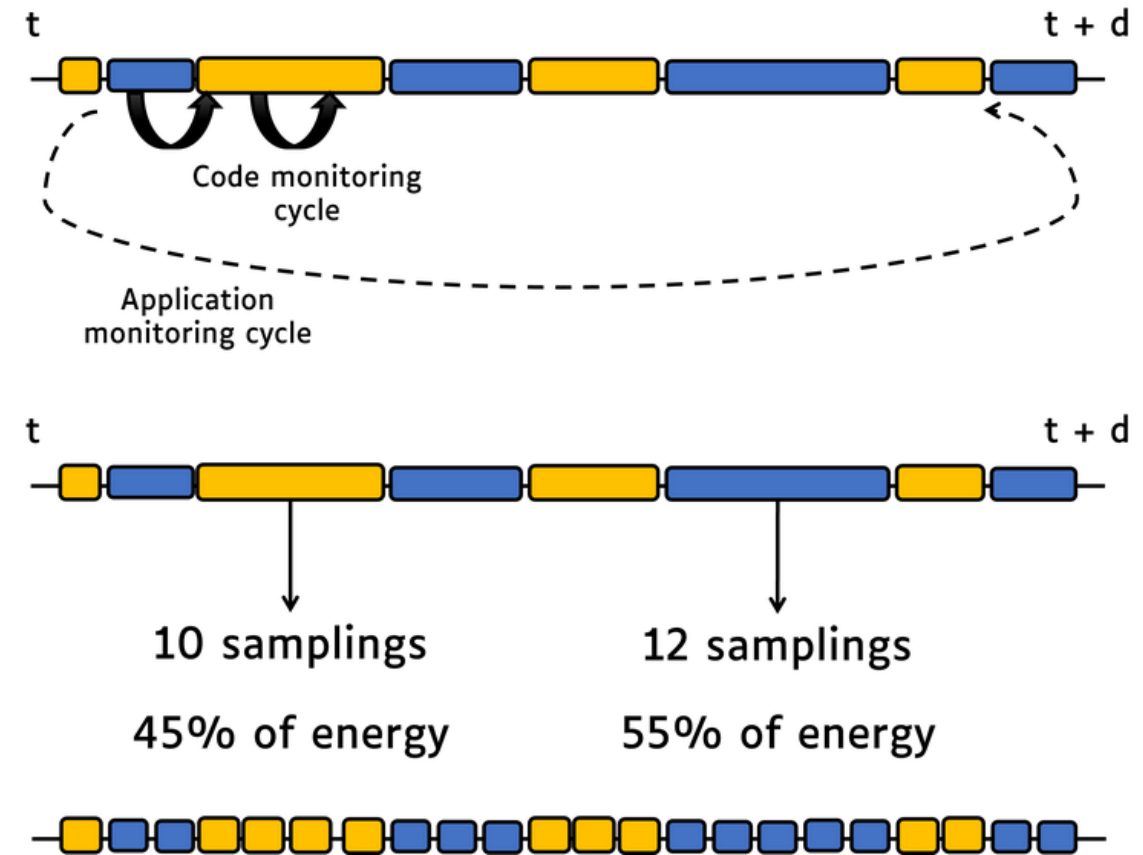
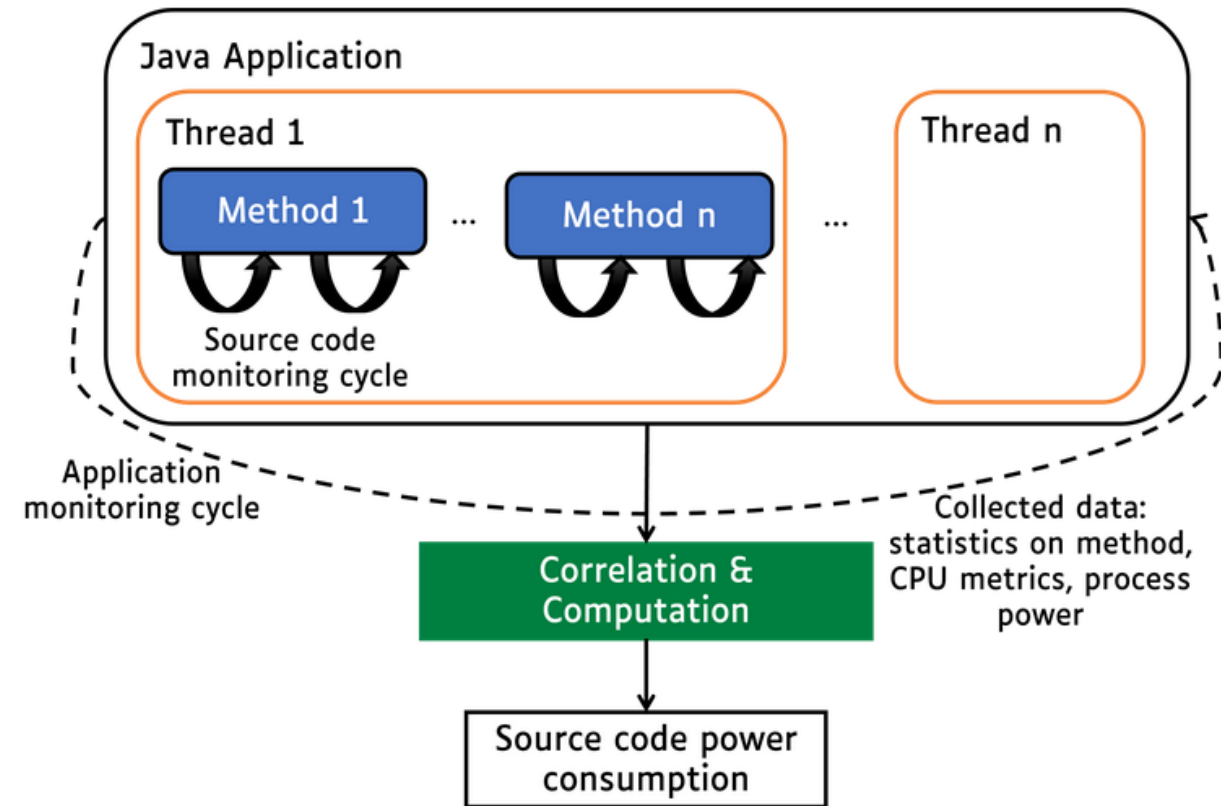


References

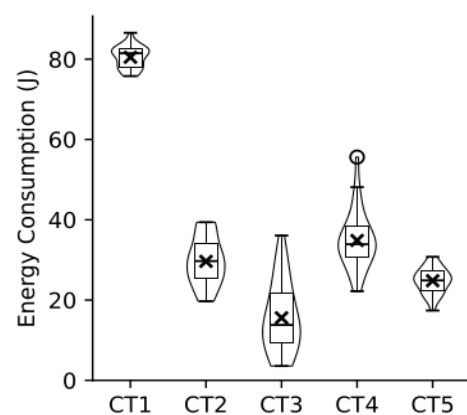
- [1] Zakaria Ournani, Romain Rouvoy, Pierre Rust, and Joël Penhoat. 2020. On Reducing the Energy Consumption of Software: From Hurdles to Requirements. In ESEM 2020 - ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM '20). Association for Computing Machinery, New York, NY, USA, 1–12. doi:10.1145/3382494.3410678
- [2] Adel Noureddine. 2022. PowerJoular and JoularJX: Multi-Platform Software Power Monitoring Tools. In 18th International Conference on Intelligent Environments. IEEE, Biarritz, France, 1–4. doi:10.1109/IE54923.2022.9826760
- [3] Adel Noureddine. (2025). How JoularJX Works—JoularJX Documentation. https://joular.github.io/joularjx/ref/how_it_works.html
- [4] Luís Cruz. 2021. Green Software Engineering Done Right: a Scientific Guide to Set Up Energy Efficiency Experiments. <http://luiscruz.github.io/2021/10/10/scientific-guide.html>. doi:10.6084/m9.figshare.22067846.v1 Blog post.

Energy consumption assessment

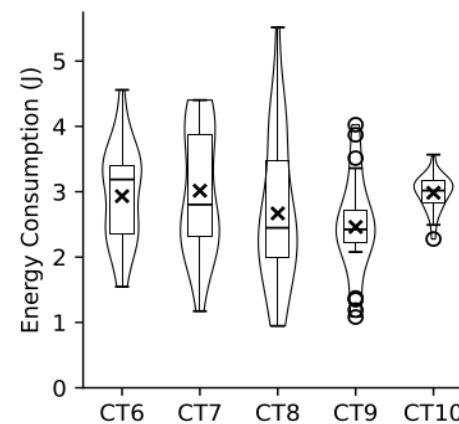
How JoularJX works [3]



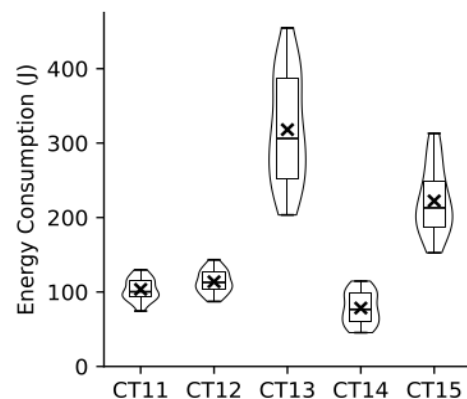
Evaluation Results



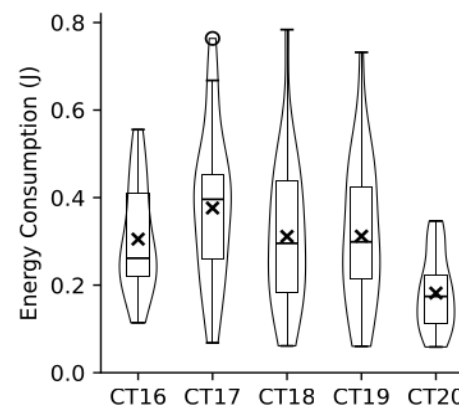
(a) Distribution of the five highest CT for Spring Boot



(b) Distribution of the five lowest CT for Spring Boot



(c) Distribution of the five highest CT for Spoon



(d) Distribution of the five lowest CT for Spoon