Software Energy Efficiency: Between Technical and Human Approaches

Dr Adel Noureddine
Associate Professor
University of Pau and Pays de l'Adour

Green Software and Human Actors: design, code, and behavior - community workshop
ICT4S 2023 - Rennes, France

5 June 2023

About me

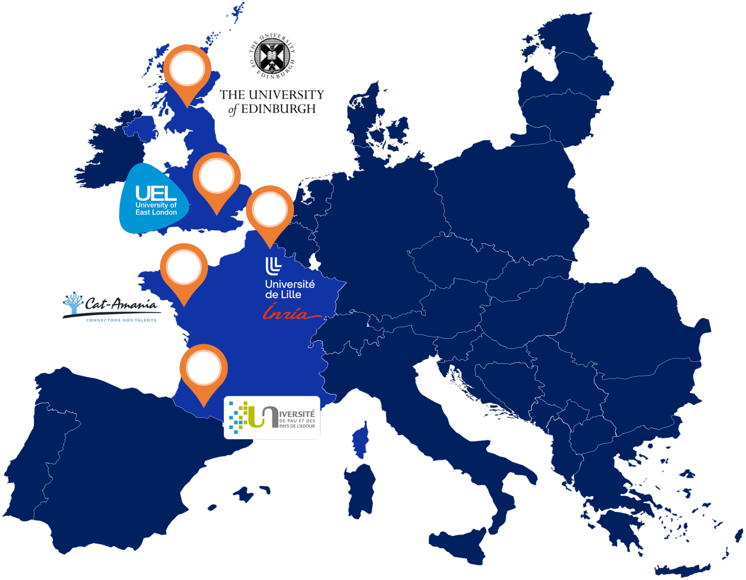
Adel Noureddine

Associate Professor in Computer Science
@University of Pau and Pays de l'Adour
@LIUPPA laboratory

Researcher in Green IT, Software Engineering and Autonomic Computing

noureddine.org

Background & Experience



What is software?

Software?

- Software is intangible (cannot be touched)
- Idea, written in text in a specific language
- Book of instructions to operate a machine
- Automated book of instructions
- Different execution contexts: PC, server, mobile, smart speaker, etc.

Software energy?

- Only physical machines consume energy (mainly electric power)
- Machines execute the instruction of software
- Energy consumed by machine to execute instructions is defined as software energy

How to measure software energy?

Energy and power

- Energy is the quantity of effort transferred to an object to achieve a certain work
- Power is the quantity of energy consumed per unit of time
- Electric power is the amount of electric energy transferred by an electric circuit

Measuring software energy

- Physical meters (power meters, sensors, power plugs, etc.)
- Software power meters (estimation models, APIs, etc.)
- Most notable APIs: Intel RAPL (Intel & AMD CPUs), Nvidia SMI (Nvidia GPUs)
- Most notable tools: PowerJoular, PowerAPI, Scaphandre, many more...
- Monitor source code energy: JoularJX, pyJoules
- Black box or white box measurements

PowerJoular: multi-platform monitoring software

PowerJoular and JoularJX: Multi-Platform Software Power Monitoring Tools

Adel Noureddine

Université de Pau et des Pays de l'Adour, E2S UPPA, LIUPPA, Anglet, France

In the 18th International Conference on Intelligent Environments (IE2022)

PowerJoular

- PowerJoular is a GNU/Linux tool to monitor power consumption of hardware and software
- Support multiple architectures: x86/64, ARM on Raspberry Pi and Asus Tinker Board
- Monitors CPU with Intel RAPL or our power models on ARM
- Monitors GPU with Nvidia SMI

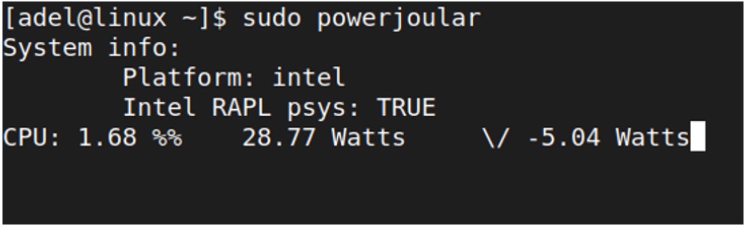
PowerJoular

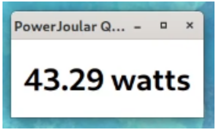
- Can monitor hardware (CPU, GPU) and software (process and applications)
- Provides a systemd service (daemon) to continuously monitor power
- Expose power consumption on runtime in terminal and CSV files
- Low overhead (Ada, compiled to native code)
- GPL 3
- noureddine.org/research/joular/powerjoular
- github.com/joular/powerjoular

Monitoring processes and applications

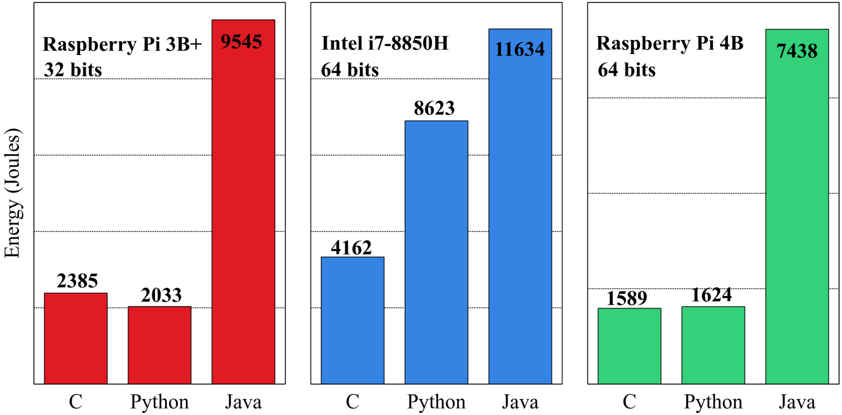
- For a process: collects CPU statistics from `/proc/stat` and `/proc/PID/stat`, to calculate CPU usage of the process every second
- For an application: every second, search all PIDs of the application (by its name), then monitor and sum their power consumption
- PowerJoular can keep up with process creation/destruction by applications

PowerJoular





PowerJoular



Energy consumption of Ray casting algorithm on different programming languages and platforms

JoularJX: source-code level monitoring

PowerJoular and JoularJX: Multi-Platform Software Power Monitoring Tools

Adel Noureddine

Université de Pau et des Pays de l'Adour, E2S UPPA, LIUPPA, Anglet, France

In the 18th International Conference on Intelligent Environments (IE2022)

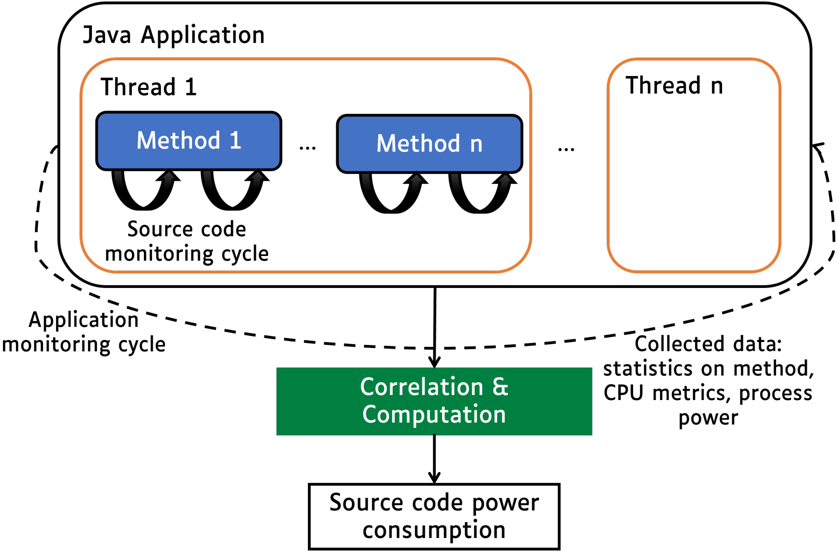
JoularJX

- JoularJX is a Java-based agent for power monitoring at the source code level
- Support multiple architectures: x86/64, ARM on Raspberry Pi and Asus Tinker Board (same approach and models as PowerJoular)
- Works on Windows and GNU/Linux
- Real time power monitoring of the source code

JoularJX

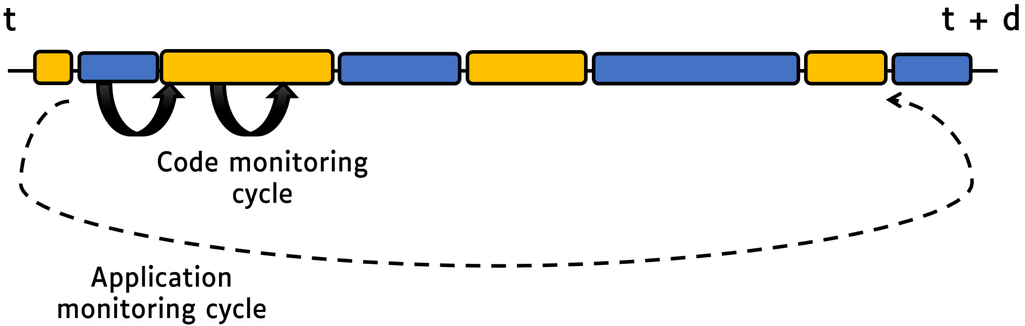
- Measures energy for every method of the application and/or the JDK
- Measures methods' call tree and branches
- Monitor power evolution of every method
- Exposes all monitored data in CSV files
- GPL 3
- noureddine.org/research/joular/joularjx
- github.com/joular/joularjx

How it works

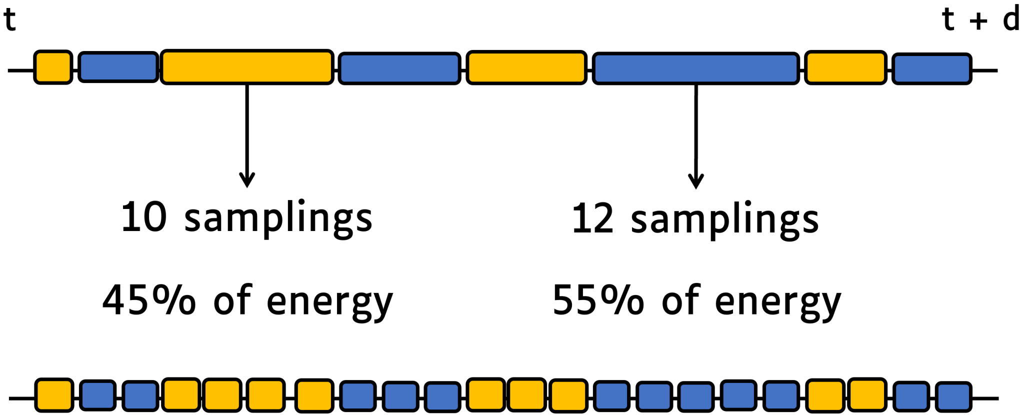


Java Application
Thread 1
Method 1
...
Method n
...
Thread n
Source code monitoring cycle
Application monitoring cycle
Collected data: statistics on method, CPU metrics, process power
Correlation & Computation
Source code power consumption

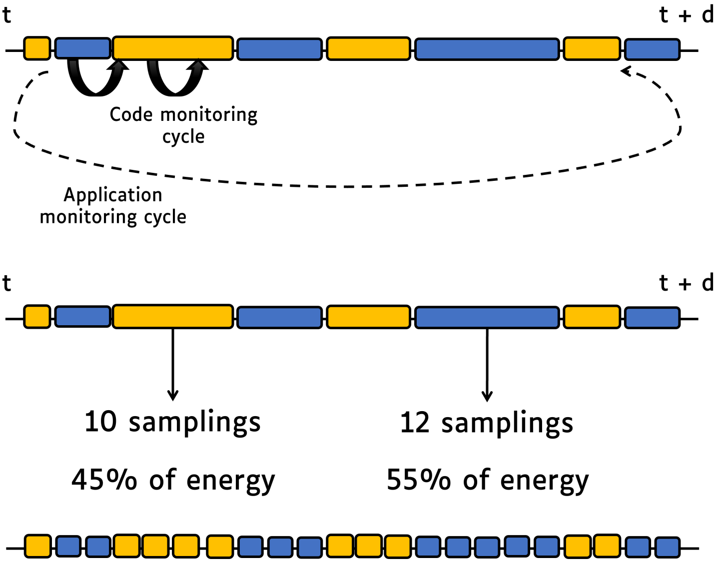
How it works



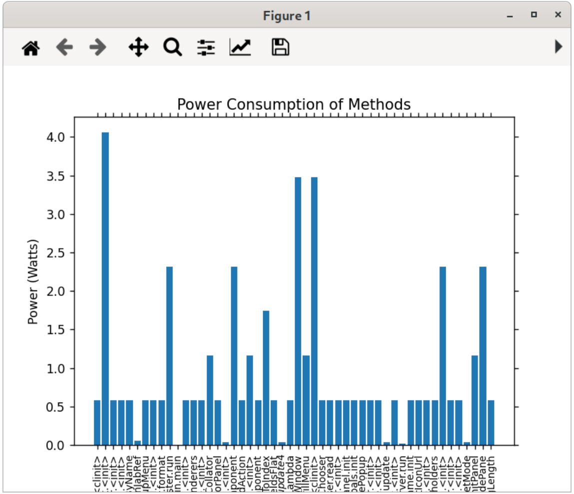
How it works



How it works

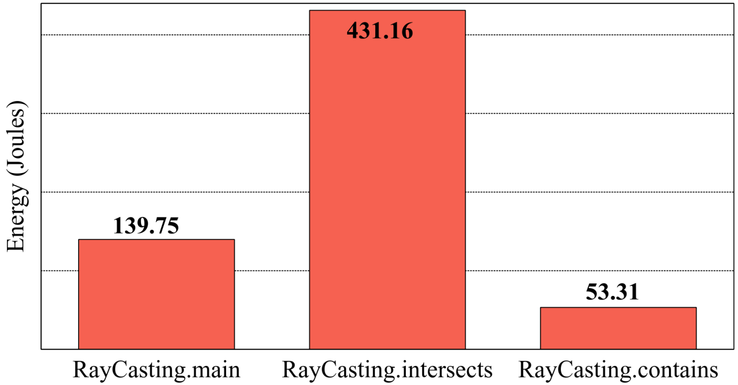

t
t + d
Code monitoring cycle
Application monitoring cycle
t
t + d
10 samplings
45% of energy
12 samplings
55% of energy

JoularJX



Example of a sample GUI

JoularJX



Energy consumption of the Java implementation of the Ray casting methods

JoularJX

```
joularjx-results
└──pid-timestamp
    ├──all
    │   ├──evolution
    │   ├──runtime
    │   │   ├──calltrees
    │   │   └──methods
    │   └──total
    │       ├──calltrees
    │       └──methods
    └──app
        ├──evolution
        ├──runtime
        │   ├──calltrees
        │   └──methods
        └──total
            ├──calltrees
            └──methods
```

Results folder structure

JoularJX: evolution through time

tion > joularJX-16224-ArrayList2.access-evolutio...

```
1675460936,1.1271
1675460938,13.2983
1675460933,6.4101
1675460997,6.1450
1675460934,8.0175
1675460998,1.6743
1675460993,1.5137
1675460928,4.2620
1675460992,2.6224
1675460931,4.1910
1675460995,13.0719
1675460930,6.4126
1675460925,2.7002
1675460988,7.7874
1675460926,10.6637
1675460990,6.9797
1675460921,18.6875
1675460985,1.9955
1675460920,1.8296
1675460984,3.1813
1675460923,0.0000
1675460987,4.3382
1675460916,8.8189
1675460918,4.9692
1675460982,0.6361
1675460913,3.2777
1675460912,5.6473
1675460915,6.4254
1675460908,5.2418
1675460910,6.1508
1675460907,4.4196
```

volution > joularJX-16224-ArrayList2.add-evolution.csv

```
1675460892,6.5584
1675460895,9.5070
1675460894,2.1506
1675460889,14.6925
1675460890,3.6545
1675460885,1.7588
1675460884,19.2151
1675460887,5.7996
1675460881,9.4096
1675460882,3.5240
1675460877,2.0728
1675460879,10.9698
1675460905,5.8707
1675460907,0.6604
1675460900,8.8889
1675460903,14.8011
1675460902,3.9778
1675460897,10.7301
1675460898,6.5253
```

volution > joularJX-16224-ArrayList2.remove-evolutio...

```
1675460957,14.0395
1675460956,16.6754
1675460959,16.5045
1675460952,21.4660
1675460954,8.8016
1675460949,62.1093
1675460948,34.0319
1675460951,42.6504
1675460944,57.4988
1675460946,62.2007
1675460941,51.4266
1675460943,56.7188
1675460939,68.8934
1675460938,22.6430
1675460980,6.4290
1675460982,7.3152
1675460977,17.0497
1675460979,6.2655
1675460972,9.1198
1675460975,7.5734
1675460974,12.8209
1675460969,12.6063
1675460970,17.2295
1675460964,5.1419
1675460967,7.8258
1675460966,22.1694
1675460961,5.2706
1675460962,29.9630
```

Power comsumption evolution of each method

JoularJX: call tree & branches

```
MainArray.main;MainArray.ArrayAccess;ArrayList2.access,49.9456
MainArray.main;ArrayList2.access,122,5503
```

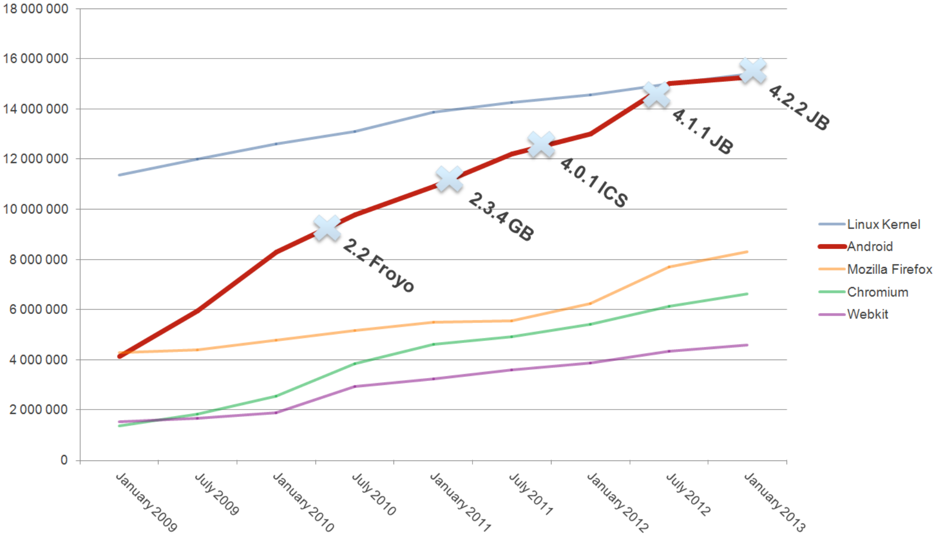
Energy consumption of the call tree and branches

Optimizing software energy

Optimizing software energy

- Source code optimizations (instructions, snippets): HashMap vs ArrayMap, reduce function calls, reduce objects creation, recursive/iterative algorithms, etc.
- Three principles of eco-design (Syntec numérique, 2013):
  - Minimize software usage time and the required resources to deliver the software workload (build efficient software)
  - Reduce the functionalities of software to its most adapted usage by users (avoid software bloatware)
  - Forecast the durability and evolution over time of the software solutions
- Reduce *feature creep* (addition of new features that go beyond the basic functionalities of an application)
- Avoid *bloatware* (software that become slower and use more hardware resources to deliver the same functionalists or suffer from feature creep than previous versions)

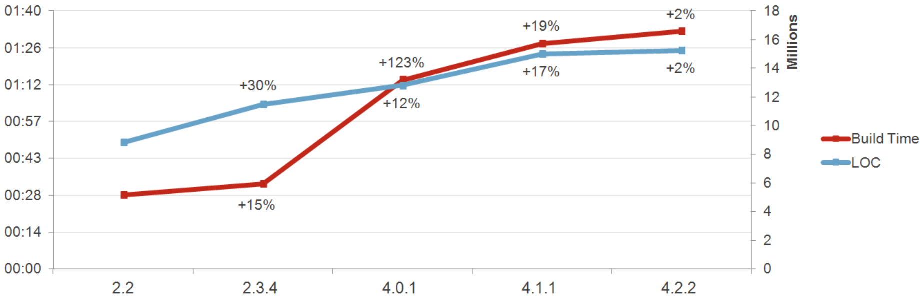
Software bloat



[1] Source: http://www.ohloh.net, http://en.wikipedia.org/wiki/Android_version_history (February 2013)

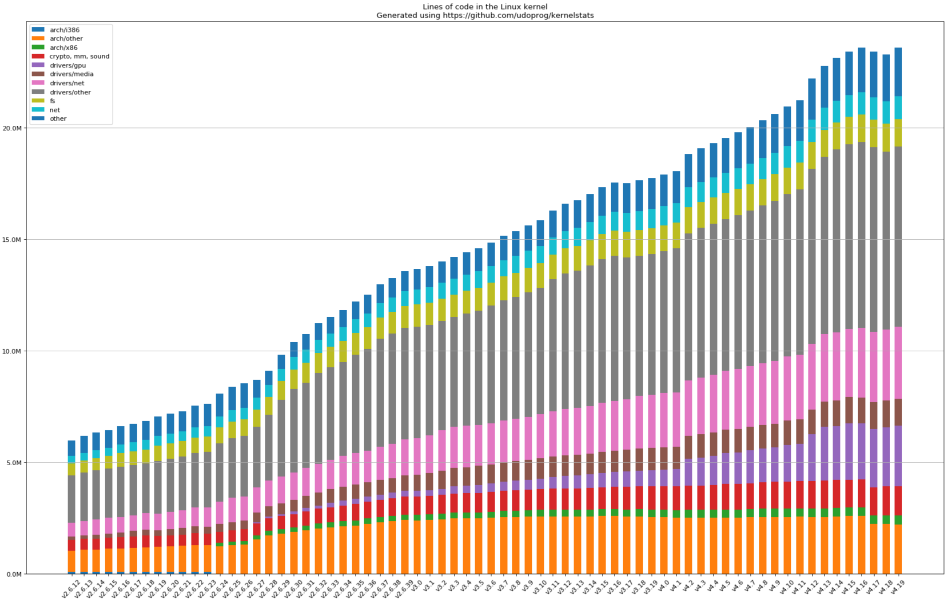
Software bloat: Android

Android Version	Release Date	LOC[1]	LOC Growth %	GNU Make Build Time[2]	Build Time Growth %
2.2	May 2010	8,837,858	-	28m55s	-
2.3.4	April 2011	11,492,324	30%	33m10s	15%
4.0.1	October 2011	12,827,330	12%	1h13m54s	123%
4.1.1	July 2012	15,028,331	17%	1h28m11s	19%
4.2.2	February 2013	15,266,803[3]	2%	1h32m56s	2%



[1] http://www.ohloh.net, [2] Builds performed on 8-core 16GB RAM server, [3] LOC data as of December 2012

Software bloat: Linux kernel



Software bloat

Studying co-running avionic real-time applications on multi-core COTS architectures, 2014

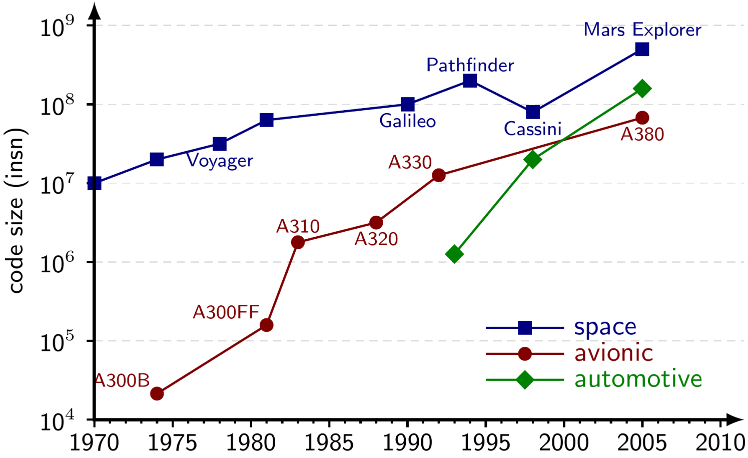


Fig. 1. Evolution of code size in space, avionic and automotive embedded systems

Reduce the usage of these resources

- Processing resources
  - Function calls which trigger CPU context switches
- Memory access
  - Number of objects created in object-oriented programming
- Network usage
  - Limit data transferred to the minimum needed
  - Limit internet connections opening (ex. open connection in a loop)
- Unused resources
  - Close database connections, network sockets, etc.

Reduce the usage of these resources

- Hardware component usage and leakage
  - Prefer network triangulation on GPS if an exact position is not needed
  - Synchronize GPS usage with other software and usages
  - Screen wake-up, bright colors on OLED screens
  - Non-core functionalities
- Non-core functionalities
  - Advertisement ("up to 75% of energy in mobile free apps are consumed by 3rd party advertisement modules" - *Fine Grained Energy Accounting on Smartphones with Eprof, 2012*)

Eco-design guidelines

- GR491 Handbook of Sustainable Design of Digital Services: `https://gr491.isit-europe.org/en/`
- Référentiel général d'écoconception de services numériques (RGESN) (in French): `https://ecoresponsable.numerique.gouv.fr/publications/referentiel-general-ecoconception/`
- The intro guide to digital eco-design: `https://eco-conception.designersethiques.org/guide/en/`
- ecoCode mobile best practices: `https://github.com/cnumr/best-practices-mobile`
- EU Code of Conduct for Energy Efficiency in Data Centres: `https://e3p.jrc.ec.europa.eu/communities/data-centres-code-conduct`
- And many more...

Green software

- Most current approaches are technical and/or dev-centric
- Humans, the ones using software, not coding, are often forgotten and outside the sustainability loop
- End-users have an important role to play
- Shifting towards user-centric sustainable software
- For example: green software feedback and behavioral change in using software (Noureddine et al., 2023)

At a crossroad...

Path ahead

- What motivates users to sustainability
- How they understand green software and its energy consumption
- Study the impact of green feedback on user behaviors
- Drive end-users behavioral changes

My current research around the topic

- Technical side: measurement tools and energy models, understand factors impacting software energy
- Human side: study the impact of green feedback, push for users behavioral change, Behave project

My current research around the topic

- Technical side: measurement tools and energy models, understand factors impacting software energy
- Human side: study the impact of green feedback, push for users behavioral change, Behave project
- The Impact of Green Feedback on Users' Software Usage. In IEEE Transactions on Sustainable Computing journal (T-SUSC). 2023.
- Behave project: noureddine.org/research/behave

About me

Adel Noureddine

Associate Professor in Computer Science
@University of Pau and Pays de l'Adour
@LIUPPA laboratory

Researcher in Green IT, Software Engineering and Autonomic Computing

noureddine.org