

Methodological Guidelines for Measuring Energy Consumption of Software Applications

Original

Methodological Guidelines for Measuring Energy Consumption of Software Applications / Ardito, Luca; Coppola, Riccardo; Morisio, Maurizio; Torchiano, Marco. - In: SCIENTIFIC PROGRAMMING. - ISSN 1058-9244. - ELETTRONICO. - 2019:(2019), pp. 1-16. [10.1155/2019/5284645]

Availability:

This version is available at: 11583/2769312 since: 2023-04-27T12:20:31Z

Publisher:

Hindawi

Published

DOI:10.1155/2019/5284645

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright

(Article begins on next page)

Methodological Guidelines for Measuring Energy Consumption of Software Applications

Luca Ardito, Riccardo Coppola, Maurizio Morisio, Marco Torchiano

Politecnico di Torino
Department of Control and Computer Engineering
Turin, Italy
name.surname@polito.it

Abstract

Energy consumption information for devices, as available in the literature is typically obtained with ad-hoc approaches, thus making replication and consumption data comparison difficult. We propose a process for measuring the energy consumption of a software application. The process contains four phases, each providing a structured deliverable that reports the information required to replicate the measurement. The process also guides the researcher on a threat to validity analysis to be included in each deliverable. This analysis ensures better reliability, trust, and confidence to reuse the collected consumption data. Such a process produces a structured consumption data for any kind of electronic device (IoT devices, mobile phones, personal computers, servers, etc.), which can be published and shared with other researchers fostering comparison or further investigations. A real case example demonstrates how to apply the process and how to create the required deliverables.

Keywords: energy consumption, software energy consumption, software application, software engineering, energy consumption data, computer engineering

1. Introduction

2 A software program contains a sequence of instructions whose execution
3 requires the device on which it is running to consume energy. Today, energy
4 consumption, a non-functional property of the program, is seldom considered
5 upfront as a non-functional requirement or, after the fact, as a property to

6 be measured and monitored. However, energy consumption may represent a
7 critical problem for end users. In laptops, tablets, and smartphones, energy
8 consumption clearly has an impact on battery life and, therefore, it becomes
9 a user experience issue [1]. For data centres or Bitcoin miners [2], energy
10 consumption has a direct impact on the electrical bill. In the literature, many
11 have addressed the problem of measuring and reducing energy consumption,
12 but typically in an ad-hoc manner [3].

13 According to the Evidence-Based Software Engineering (EBSE) [4] ap-
14 proach, concrete decision-making should be supported by the empirical evi-
15 dence available in the literature. Such evidence must be trustable, produced
16 through a documented and repeatable process, contextualised, linked to the
17 context where it can be applied, identifiable, address a well-defined question,
18 assessable, and report the known limitations of the results. Such character-
19 istics are seldom present in most of the related published literature.

20 If the energy consumption issue is tackled at the hardware level, then the
21 task is accomplished by reducing the consumption of the physical devices
22 or by creating different usage profiles (e.g., processors can scale down the
23 frequency when used less). On the other hand, if the energy consumption
24 issue is managed at the operating system level, then management policies
25 may use the different hardware profiles of various devices (when available)
26 or turn off hardware when not needed. We consider software as a driver of
27 the energy consumption because it requires several actions to be completed
28 by the underlying hardware, which reacts based on the received instructions.
29 Measuring the energy consumption due to a specific piece of software implies
30 addressing two major issues:

- 31 • Isolating the energy consumption of a program when it is running con-
32 currently to others on the same device.
- 33 • Generalizing the obtained results: let the measure be meaningful to
34 other devices.

35 When collecting energy through physical measurements on a device, the
36 value is related to the target software and all other processes running on the
37 device simultaneously. The physical measurement does not allow a straight-
38 forward generalisation of results because the same software could behave
39 differently based on the hardware on which it is executed as well as other
40 installed software. Another option is defining models that provide an esti-
41 mate of the energy consumption of the target software instead of performing a

42 physical measurement. The input of the model consists of device resource us-
43 age indicators collected at run-time. The main issue affecting this approach
44 is that a model can be representative of a device or a family of devices
45 meaning that the estimation computed by the model is not always valid.
46 Unfortunately, it is very difficult to get this result because every hardware
47 manufacturer should provide accurate data on the consumption of the device,
48 and this data should be available in real time as device status information
49 through sensors and system calls from the operating system. Now, we can
50 easily measure the energy consumption of an application by measuring the
51 energy consumption of the entire device on which it is executed, analysing
52 the obtained data, and estimate the consumption by minimising the error.
53 This requires a precise methodology to obtain the most significant data and
54 analyse them for useful information to estimate the power consumption of
55 an application.

56 In this paper, we propose a general process that can be used to measure
57 the energy consumption of a software application. This process includes
58 the best practices for collecting and analysing energy consumption data of
59 a software application and formalises the steps needed to carry out a valid
60 empirical experiment. Thus, this is proposed as a ground zero for performing
61 software energy measurements to ensure repeatability and comparison of each
62 experiment. The process we put forward can be used both to conduct energy
63 measurement and to assess existing studies serving as a sort of checklist.

64 The remainder of this paper is organised as follows:

- 65 • Section 2 describes the proposed process to collect energy consumption
66 data from devices as well as how to analyse the data;
- 67 • Section 3 provides a real case study showing how to create the deliver-
68 ables;
- 69 • Section 4 reports the related work and assess the literature in terms of
70 compliance with our process;
- 71 • Section 5 concludes the manuscript and provides hints for future work.

72 **2. Software Consumption Measurement Process**

73 The proposal described in this paper is a repeatable process for measuring
74 the consumption of a software application, hereinafter called the Software
75 Under Test (SWUT). The process consists of the following four phases :

Table 1: Deliverables of the different phases and impact on threats to validity

Phase	Input	Output	Threats to Validity
Goal	-	SWUT and Context Research Questions Devices	External : Generalization of results
How	Goal Deliverable	Instrumentation Synchronization Sampling File Format	Internal : Assigning consumption value to a process Construct : Incorrect Measurement Conclusion : Insufficient number of repetitions
Do	How Deliverable	Measurement Scripts Data Files	Construct : Incorrect Implementation
Analyse	Do Deliverable	Data Analysis Scripts Results and Discussion	Conclusion : Not suitable statistical tests

- 76 • Goal (G): define the research question, the target device(s) on which
77 the measurements will take place, and the context in which the SWUT
78 is executed.
- 79 • How (H): decide how consumption will be measured and the procedure
80 needed to carry out the measurement.
- 81 • Do (D): carry out the measurement and collect the data.
- 82 • Analyse (A): analyse the data and address the research question(s).

83 The UML Activity Diagram in Figure 1 summarizes the main activities
84 and decisions encompassed by the process and the relative threats to the
85 validity of the results.

86 Each phase of the process shall produce a deliverable, which summarises
87 the decisions taken, the outcomes of the phase, and the said analysis of the
88 threats to validity. A summary of the elements provided by each deliverable
89 is provided in Table 1. As it is evident in the table, each deliverable serves
90 as an input for the following one.

91 The following subsections describe each phase of the process along with
92 the required information to reproduce it.

93 A sample application of the described process to a simple case study will
94 be then described in Section 3.

95 Each phase requires a few decisions to be taken, some of which can in-
96 fluence the validity of the results. Wholin et al. [5] classified the threats to

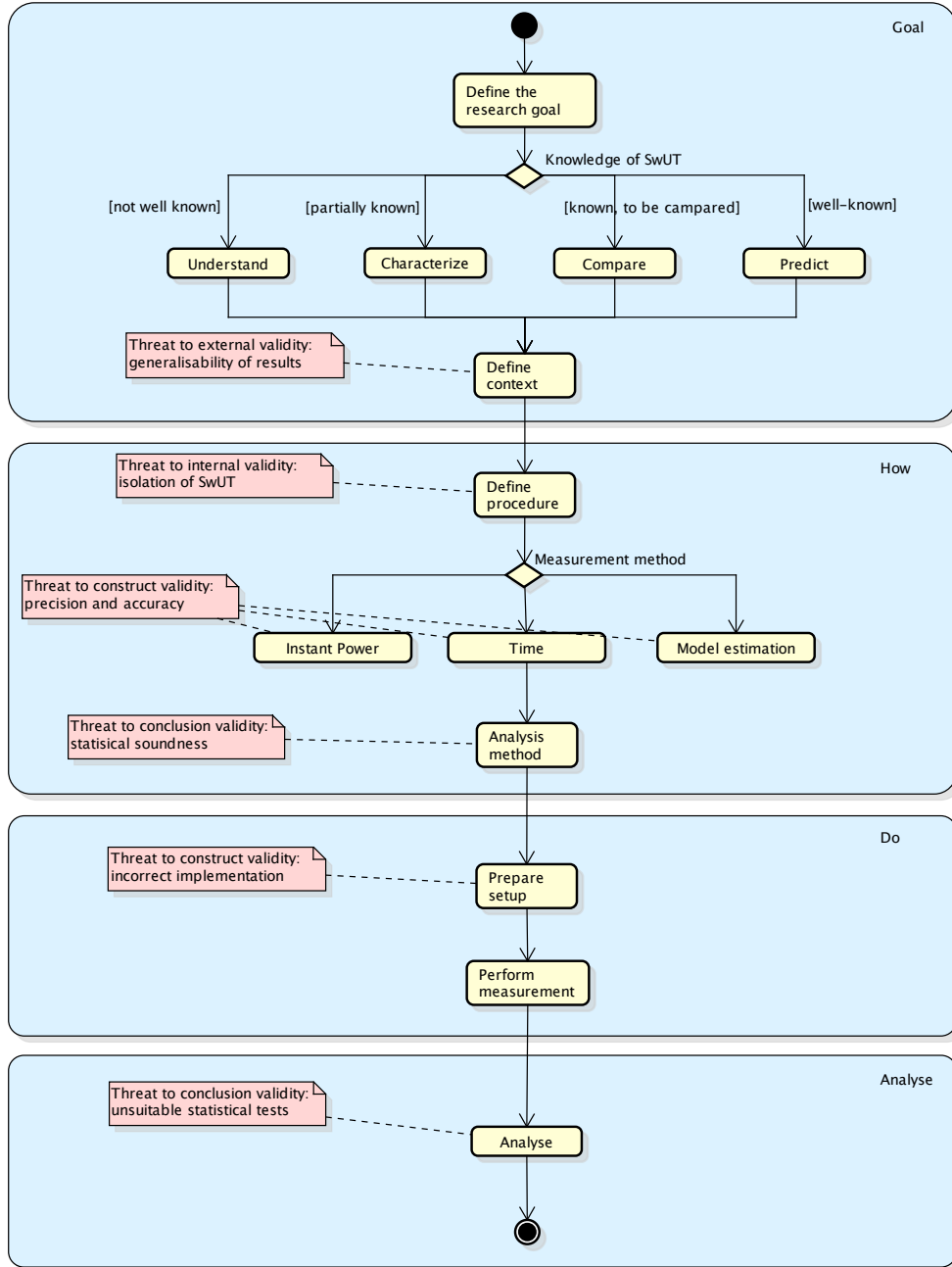


Figure 1: Summary of activities and decisions of the proposed process.

97 validity as:

- 98 • Internal Validity: focused on how sure it is that the treatment actually
99 caused the measured outcome;
- 100 • Construct Validity: focused on the relation between the theory behind
101 the experiment and the observation;
- 102 • External Validity: focused on the generalizability of the results outside
103 the scope of the study;
- 104 • Conclusion Validity: focused on the relationship between the treatment
105 used in the experiment and the actual outcome measured.

106 Table 1 shows these categories of threats and how they are impacted by
107 each phase of the process.

108 *2.1. Phase I: Goal*

109 This phase is about defining the research questions that will drive the
110 measurement process. Since the scope of the research questions is restricted
111 to energy consumption, we propose to represent the goal as a template in-
112 spired by the GQM approach [6]:

113 “< understand | characterize | compare | predict > the <consumption>
114 of the <SWUT> run on <device(s)> in <context(s)>”.

115 An example of a research question obtained applying this template is:

116 *“Characterize the energy consumption of the Bubble Sort algo-
117 rithm implemented in Java language when run on Raspberry Pi
118 version 2B in the context of Raspbian Linux OS”.*

119 The first aspect to consider is the purpose of the measurement, which
120 depends on the level of knowledge of a specific process, and includes the
121 following options:

- 122 • Understand: this goal applies to the initial investigations for a process
123 that is not well known to understand the input and output variables of
124 the process. Nominal or ordinal measures may characterize variables.

- 125 • Characterize: this goal applies to a process that is partially known
126 to enhance its description by providing the input, output, and context
127 variables that influence the process. Interval, ratio or absolute measures
128 may characterize variables. Relationships between the variables, either
129 analytic or probabilistic, are proposed, but their validity is limited.
- 130 • Compare: this goal is a variant of characterising where two similar
131 SWUTs are characterised and compared on the variables defined.
- 132 • Predict: this goal applies to well-known processes to provide a model
133 that relates all variables in the process. The validity of the model is
134 broad, so that output variables are predicted by input variables reliably.

135 Consumption can be measured in terms of energy [*Joule*] or power [*Watt*],
136 which are related and one may be computed from the other. However, in
137 practice they are not entirely interchangeable:

- 138 • From the research goal, power offers an immediate view and is suitable
139 for tasks with a very long (possibly infinite) duration, while energy is
140 suitable for tasks with a finite duration. For instance, if the software
141 function to be measured is “read an email message”, or “convert an
142 audio file from mp3 to wma”, then energy is the most suitable to char-
143 acterise the consumption of the functions. If the software function is
144 “control the speed of an engine”, then power is the most suitable.
- 145 • From the measurement as a function of the hardware configuration
146 (server vs desktop vs mobile phone), it may be way easier to measure
147 power compared to energy, which will be discussed in Section 2.2.

148 The SWUT can represent a function, a set of functions, a software pro-
149 cess, a software application or a software application subset of features. The
150 description of the SWUT includes the programming language, the toolchain
151 used to produce it (e.g., the compiler and its version or the linker and its
152 version), and the usage scenario. Harman and colleagues [7] identified three
153 levels of SWUT granularity: fine grained, corresponding to individual lines of
154 code or statements; mid-grained, that is a block of code or a method/proce-
155 dure; or coarse-grained addressing a whole program execution over a period
156 of time.

157 The device represents the physical device (or devices) specifications (make,
158 model, version, CPU, architecture, and memory) used in the experiment.

159

160 The context describes other attributes that may influence the experiment,
161 such as:

- 162 • The operating system;
- 163 • The list of processes running while the measurement is performed;
- 164 • The device configuration;
- 165 • Any hardware and software instrumentation used to collect the energy
166 information.

167 Since a SWUT can be very complex, addressing the research questions
168 may require the creation of many subgoals, which aim at measuring the en-
169 ergy (or power) consumption of a predefined subset of features of the complex
170 SWUT. We will provide a complete example in Section 3.

171 As seen in Table 1, the decisions must consider the threats to external
172 validity, which regard the generalisation of results:

- 173 1. Threats help identify whether the results are valid only for the analysed
174 device(s) and context(s) or they have wider validity.
- 175 2. Threats define the importance the obtained results will be valid on
176 other devices or contexts. If yes, then researchers should state how
177 device(s) and context(s) should be selected to minimise the external
178 threats to validity. If not, researchers should state if it is in the goal of
179 the experiment to obtain results only for a specific device and context.

180 This type of analysis during the early stages of the process has a twofold
181 contribution. It makes the experiment more precise and formal as well as
182 forcing who is experimenting to choose the best context(s) and device(s) to
183 reach the goal.

184 The output of this phase is a deliverable which contains the goal descrip-
185 tion comprised of research question(s), device(s), SWUT, context, and the
186 external threats to validity analysis.

187 *2.2. Phase II: How*

188 With the unit of measure (energy or power) determined in the first phase,
189 this step will decide how to take the measurement. The three options are de-
190 scribed in the following, whereas Table 3 analyses the benefits and drawbacks
191 of each technique.

192 *Instant Power Measurement.* This technique measures the instantaneous cur-
193 rent consumed by the device and then multiplies this value by the voltage.
194 The integral over a period gives the energy value. Instant power measure-
195 ments are precise if the sampling frequency is high, but they require physical
196 instrumentation. This approach usually operates at the device level – al-
197 though hardware component-level measurement is possible – and can work
198 with coarse grained SWUT only.

199 *Time Measurement.* Another way to collect the energy consumption of a
200 device is through measurement of time. A fully charged (and healthy) battery
201 holds a known amount of energy (e.g., 1000 mAh corresponds to 18 kJoules).
202 Assuming a constant consumption over time, the speed at which energy is
203 depleted depends on the power consumption of the device. So, the average
204 power consumed is computed by measuring the time to discharge the battery
205 completely. This measurement relies on the precision of the battery capacity
206 measure. If this value is imprecise, then so will be the calculated consumption
207 value. Another issue is how linearly the battery discharges, especially if
208 a measure is collected without fully discharging the battery. For devices
209 without a battery (e.g., SoC computers, such as a Raspberry Pi), the type
210 of measurement is possible by connecting the device to a battery instead of
211 connecting it to the electrical network. This approach has the same limitation
212 as the previous one in terms of granularity.

213 *Model Estimation.* Consumption measurements through models are calcu-
214 lated in a way that relates the power consumption of a particular device
215 with internal resource usage indicators, such as the CPU states, instructions,
216 memory or disk accesses, and network adapters. In the literature, there exist
217 few examples of power models. For example, A. Patak et al. [8] described a
218 power module based on system call tracing. This approach uses system calls
219 for estimating the resource usage. Di Nucci et al. [9] proposed a software-
220 based approach, named PETRA, proving that those methods are not in-
221 herently less precise than hardware-based or model-based solutions. Their
222 approach is specifically aimed at testing Android applications. A. Nacci et

Table 2: Elements of the How deliverable

	Instant Power	Time Measurement	Model Estimation
Hardware Instrumentation	✓	✓	-
Software Instrumentation	-	✓	-
Synchronization	✓	-	✓
Sampling Frequency	✓	-	✓
File Format	✓	✓	✓
Threats to Validity	✓	✓	✓

223 al. [10] introduced an approach to build a power model for Android devices
 224 by using Android APIs to retrieve a variety of states, including the battery,
 225 network connection, Wi-Fi, and screen. Two components usually implement
 226 the models:

- 227 • A resource usage analyser that measures the usage of resources on a
 228 computer, which depends on the operating system;
- 229 • A resource usage to consumption converter that reads the data pro-
 230 vided by the resource usage analyser and, based on the mathematical
 231 model, it converts to consumption values. The mathematical model is
 232 a parameter that varies according to the device.

233 The latter component requires choosing a model suitable for the device on
 234 which the SWUT will run. The model should provide the estimation error,
 235 the sampling frequency at which the resource usage is updated, and the
 236 overhead caused by extracting the resources utilisation. The overhead is a
 237 crucial value because a software process implementing the model executes the
 238 resource usage data collection, and, as with all the other software processes,
 239 it affects the consumption of the device on which it is executed. The sampling
 240 frequency and overhead are directly proportional.

241 This latter approach has the advantage of being applicable also to a fine
 242 grained SWUT.

243 The output of this phase is called *How Deliverable* as described in Table
 244 1, which contains the key decisions used for obtaining the consumption of the
 245 SWUT. The deliverable will contain different elements based on the selected
 246 measurement approach, as shown in Table 2.

Table 3: Evaluation of measurement techniques

Measurement Technique	PROS		CONS	
	Energy	Power	Energy	Power
Instant Power Measurement	Precise if sampling frequency is high	-	Physical Instrumentation needed. Difficult to isolate a single software application's contribution.	
Time Measurement	Precise if the exact energy stored in the battery is known	-	Requires many repetitions of long tasks. Difficult to isolate a single software application's contribution	
Model Estimation	No instrumentation required. Easy to isolate a single software application's contribution		Precision not always declared	

247 The components of the How deliverable, and the way they vary according
248 to the selected approach, are detailed in the following subsections.

249 2.2.1. Hardware Instrumentation

250 Hardware Instrumentation is required by the approaches based on Instant
251 Power and Time Measurement.

252 *Instant Power.* To perform power measurement, the following hardware in-
253 strumentation is required:

- 254 • A Voltage Generator;
- 255 • A shunt resistor (e.g. 0,05 Ω);
- 256 • An ADC (analog-to-digital converter);
- 257 • A supervising device.

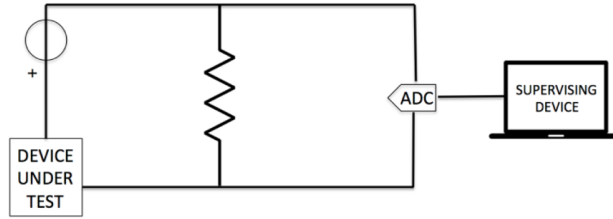


Figure 2: Circuit designed to measure instant power consumption

258 Figure 2 shows a typical configuration to measure instant power consumption
 259 data from the device. An ADC reads the voltage drop V across the shunt
 260 resistor. This data is sent to the supervising device, which will be later used
 261 for the analysis. According to Ohm's Law, V/R provides the current I , so
 262 the instant power consumption is calculated by $P = V \cdot I$. If the device has
 263 a battery pack, it should be removed because the voltage generator will also
 264 charge the battery pack during the experiment, providing inconsistent values
 265 to the ADC. Uncertainty on the power is $u(P) = P * (u(V)/V + u(I)/I)$.
 266 Both uncertainties are due to measurement errors and are typically relatively
 267 small when using suitable devices. On the market there are several power
 268 meters that can be used for the different categories of devices (e.g. mobile
 269 phones or single board computers, PCs, etc.). It is not required to build a
 270 power meter, however, its internal structure can be simplified to the circuit
 271 described in Figure 2.

272 *Time Measurement.* As described in Figure 3, a supervising device takes the
 273 system times during the test run, when the battery level changes, and when
 274 the device battery is completely discharged. For automating the time mea-
 275 surement, a programmable switch (represented by the dotted line connection
 276 between the supervising device and the switch) may be used to manage the
 277 charging process of the battery when it reaches a predefined discharge value
 278 (e.g., 2%). If the battery information is not available, then the predefined dis-
 279 charge value is 0%, and the device under test will turn off. Here, the problem
 280 is how to trigger this event. An example could be reading the output voltage
 281 value of a USB port with an ADC. When the voltage starts decreasing, the
 282 device is turning itself off, so this event can trigger the battery recharge.

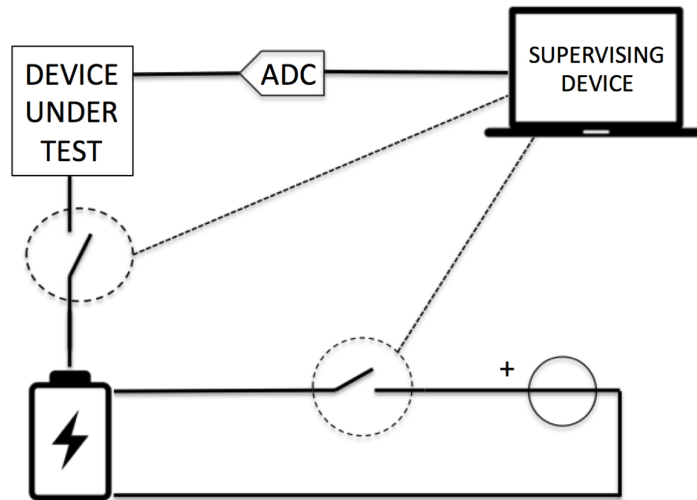


Figure 3: Hardware configuration for time measurement

283 *2.2.2. Software Instrumentation*

284 The Software Instrumentation is required only if the Time Measurement
 285 approach is selected. Time measurements require an automated procedure,
 286 which calls the SWUT continuously until the battery is discharged. At the
 287 end of the measurement, the result is an average consumption of the entire
 288 test run. To summarise, the measurement procedure should perform the
 289 following steps:

- 290 • Charge the battery until maximum battery level;
- 291 • Record the system time;
- 292 • Run the SWUT inside a loop until the battery is completely discharged.
 293 Typically, the SWUT is not able to completely discharge the battery
 294 in a single execution, so it must be run many times in an infinite loop
 295 while recording the number of runs;
- 296 • Record system time when the battery charge level changes (if this data
 297 is available);
- 298 • Re-record the system time when the battery charge level reaches a pre-
 299 defined minimum value or when it is completely discharged. Compute
 300 the experiment total time T and the number of runs, and then store
 301 the results in a file;

- 302 • Recharge the battery until it is fully charged;
- 303 • Repeat these steps to obtain reasonable statistics (e.g., 30 data points
- 304 represents a meaningful dataset [11]).

305 Once the raw data is collected, the average power consumption is computed
306 by analysing the time spent to completely discharge the battery as $\bar{P} =$
307 $(C/T) \cdot V$, where:

- 308 • \bar{P} is the average power consumption consumed in an hour,
- 309 • C is the total capacity of the battery in mAh¹,
- 310 • T is the time needed to discharge it in hours, and
- 311 • V is the voltage provided by the battery.

312 While the total energy consumed can be computed as $E = C \cdot V$.

313 The uncertainty on the \bar{P} is $u(\bar{P}) = \bar{P} * (u(C)/C + u(T)/T + u(V)/V)$,
314 thus depending on:

- 315 • $u(C)$: the uncertainty on the actual battery capacity, this is the most
- 316 critical since battery tend to change their capacity over time and even
- 317 new batteries might have actual capacity quite different from the nom-
- 318 inal one;
- 319 • $u(T)$: the error in the time measurement: this error is typically small
- 320 since complete battery discharge requires a long time;
- 321 • $u(V)$: the error in the voltage measurement: this error must be mini-
- 322 mized using suitable measurement devices.

323 This technique assumes a constant power consumption value over the
324 entire battery discharge time.

¹Or the totale capacity minus the residual capacity at the predefined minimum.

325 *2.2.3. Synchronization*

326 *Instant Power.* In this approach, the consumption data – collected with a
327 certain sampling frequency – is available on the supervising device used for
328 collecting the data. However, power consumption must be associated with
329 the process executing the SWUT, and this information is available on the
330 DUT. In other words, it is needed to synchronise the time scales of the DUT
331 and the supervising device. This problem can be solved in two ways:

- 332 • Synchronize the DUT and the supervising device system times so that
333 each sample belongs to a known timestamp.

- 334 • Instrument the code by adding distinctive power patterns for a defined
335 period before and after each run.

336 The first approach requires accurate time synchronisation between the DUT
337 and the measurement device to record only the consumption related to the
338 SWUT execution. The synchronisation could be achieved using NTP (Net-
339 work Time Protocol). However, this solution can cause errors of more than
340 100 ms due to network congestion. It also requires both the supervising de-
341 vice and the device under test to be connected at least to a LAN to reach
342 the NTP server. An error in the synchronisation between the two devices
343 can lead to data invalidation, especially in experiments carried out in cas-
344 cade because the consumption data collected is not entirely related to the
345 SWUT. The second approach allows for the association of the consumption
346 to a SWUT without synchronisation by adding markers in the SWUT. The
347 markers are known as code patterns, which produce distinctive data con-
348 sumption patterns identifiable by data analysis after the data collection, and
349 may be defined as:

- 350 • Busy Marker: a function executing an empty infinite loop.

- 351 • Sleep Marker: a call to the sleep function.

352 It is possible to automatically identify these well-known patterns in the con-
353 sumption data using signal processing techniques because the busy marker
354 has very high power consumption, while the sleep marker has very low power
355 consumption (see section 2.4). Figure 4 presents three busy markers, two
356 sleep markers, and one execution of the SWUT tagged as work.

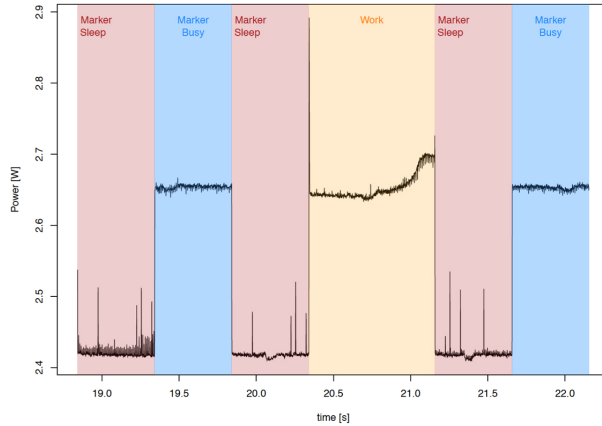


Figure 4: Consumption data and instrumented SWUT

357 *Model Estimation.* The problem related to time and data synchronisation
 358 is similar when this approach is adopted. Instead of having a consumption
 359 value, there will be resource usage data. It will then be required to translate
 360 the resource usage data into consumption data allowing the use of a times-
 361 tamp to isolate the consumption data related to the SWUT. Alternatively, it
 362 is possible to add a marker before and after the SWUT to identify the SWUT
 363 consumption data between the two markers. While the latter approach can
 364 be followed exactly, the first approach is more straightforward because the
 365 model estimation does not require a supervising device, and hence there is no
 366 need to perform clock synchronisation. So it is possible to isolate the SWUT
 367 consumption data by using timestamps.

368 2.2.4. Sampling Frequency

369 A sampling frequency is required when the Instant Power and Model Es-
 370 timation approaches are adopted. The instant power consumption measure-
 371 ment represents the average power consumption in each sample. A suitable
 372 sampling frequency is 125 kHz because only 1% of energy is consumed above
 373 this frequency as stated by Saborido et al. [12]. The authors stated that a
 374 10 kHz measurement could lead to an error of 8%, so such a low sampling
 375 frequency causes significant errors.

376 The size of the data log should be considered as another constraint. Con-
 377 sidering that each sample could be ~ 10 bytes, at 10 kHz frequency the script
 378 produces ~ 100 Kbytes per second. So, the sampling frequency should be
 379 selected carefully based on the duration of the process running the SWUT,

380 the related size of the data logged, and the acceptable error.

381 The same sampling frequency tradeoffs are valid for the Model Estimation
382 approach. However, it should be taken into account that logging the resource
383 usage too frequently can cause a sensible overhead.

384 2.2.5. File Format

385 The How Deliverable document contains the explanation of the raw data
386 file format that is used.

387 For Instant Power measurements, the raw data is included in a plain text
388 file with each line containing the instant current in A in the sample time T.
389 This format is simple, easy to read, and does not contain any extraneous data.
390 If the instant measurement contains multiple data (e.g., current, voltage, and
391 the current system time), then it is better to organise the file in JSON or
392 XML format to explicitly express the type of data included in the file. Such a
393 definition of a file format and content is useful for creating data file parsers.
394 The same file format can be applied to the Model Estimation approach,
395 given that the output provided by the model is parsed and converted into
396 consumption.

397 In case of Time Measurement, the raw data is included in a plain text file
398 representing the duration of the experiment. This format is simple, easy to
399 read, and does not contain any extraneous data, for example, 1:15:13.041454.

400 2.2.6. Analysis method

401 The typical goal of an energy measurement campaign is to assess whether
402 any main factor, e.g., a specific algorithm or computation architecture, affects
403 the energy consumption for specific tasks. In addition, often the experimental
404 design allows for the monitoring of possible confounding factors. For this
405 purpose, a basic analysis approach consists of fitting a linear model for the
406 factors with the form:

$$Energy = c_{MF} \times MainFactor + c_{CF} \times CoFactor$$

407 The factor variables can be a basic indicator or continuous variables. The
408 linear model will be subject to an ANalysis Of VAriance (ANOVA) to under-
409 stand the statistical significance of the factor effects on the power. ANOVA
410 is a statistical method to analyze the difference of means among different
411 groups; ANOVA attributes the variance of means to different sources and
412 evaluates the probability that an observed difference is due to an actual ef-
413 fect of factor versus random effects (e.g. measurement noise). Typically such

414 probability – called p-value – is compared against a predefined threshold
415 (5% is a common choice) to decide whether it is possible to state that the
416 treatment had a real effect.

417 The ANOVA is a parametric test, meaning that its results are reliable
418 when a few conditions are met, the most important being the normality of
419 the samples. The normality can be checked by means of the Shapiro-Wilk
420 test; if the test return a a p-value smaller than a given α level it is possible
421 to conclude that the data is not drawn from a normal distribution.

422 When the parametric assumption for ANOVA are not be met, a per-
423 mutation test alternative to ANOVA can be used (e.g., using the lmPerm
424 R package [13]). In addition to the statistical significance, it is important
425 to evaluate the magnitude of the effect of the factors. A basic assessment
426 can be performed by looking at relative values of the estimated regression
427 coefficients or by means of standardised coefficients, such as η^2 .

428 When a simple comparison of two samples is required, without any co-
429 factor involved a t-test can be applied, being a simplified version of an
430 ANOVA.

431 2.2.7. Threats to Validity Analysis

432 Regardless of the chosen approach, the How deliverable must contain an
433 analysis of three different threats to validity.

434 *Internal validity.* It depends on whether the consumption data is related to
435 the execution of the SWUT. Several possible cases include the following:

- 436 • The device has no operating system and executes only the SWUT. The
437 consumption of the device can be attributed entirely to the SWUT.
- 438 • The device has a multitasking OS. The SWUT and other processes (at
439 the application or OS level) run concurrently. The problem is how to
440 attribute the consumption of the device to each process (and to the
441 SWUT, in particular). An option is to stop all processes except the
442 one that executes the SWUT. This is unfeasible in most OSs, so the
443 remaining option is to minimise the set of running processes to those
444 strictly required by the OS. Then, it is possible to measure the device
445 consumption both when the device is idle, i.e., only OS-related pro-
446 cesses are running, and when it is running the SWUT. The difference
447 between the two consumption values represents a reasonable approxi-
448 mation of the effective consumption attributable to the SWUT.

- 449 • The device has a multicore processor. The SWUT can be executed on
450 any core at a specific CPU frequency. For this reason, it is unlikely
451 that two consumption measurements for the same SWUT performed
452 on the same device provide the same value.

453 The execution of the SWUT not in isolation, might be less a threat when the
454 goal of the process is to perform a comparison. In such a case, a comparison
455 can be performed when assuming the noise produce by other programs is
456 similar for all tested alternatives.

457 *Construct validity.* It depends on how consumption is measured as well as
458 the precision of the measurement:

- 459 • Instant power consumption has precision impacted mostly by the pre-
460 cision of the current measurement, and by the noise produced by pro-
461 cesses executed in parallel with the SWUT (see discussion above on
462 internal validity).
- 463 • Time measurement has precision impacted by the measure of the energy
464 contained by the battery, by the non-linear discharge pattern, by the
465 reduction of battery capacity with the recharging cycles, and by the
466 time required to identify that the energy contained by the battery falls
467 below a defined threshold.
- 468 • Model estimation builds on the precision of the model as its key at-
469 tribute. The model may or may not consider relevant factors (for in-
470 stance, heating) and, therefore, produce poor estimates.

471 *Conclusion validity.* It is the final category of threats to analyse during this
472 phase. For gaining statistical evidence, the researcher must plan a certain
473 number of repetitions of the same consumption measurement. Sometimes
474 this can be an issue, especially in time measurements where each run can
475 last hours. Thus, when it is not feasible to plan many repetitions of the
476 same run, the investigator should consider a tradeoff between the number of
477 repetitions and the possible error in the conclusions. Appropriate statistical
478 tests should be used to determine the likelihood of observed differences or
479 the confidence intervals associated with measurements.

480 *2.3. Phase III: Do*

481 In this phase, the researcher implements the experiment designed in the
482 previous phases. The crucial part is the procedure automation. Each exe-
483 cution of the procedure should be autonomous, and at the conclusion, the
484 researcher should be able to collect the data without interventions. Human
485 intervention will alter the procedure execution because it will not be repeat-
486 able with the same actions. Achieving this requires defining a script that
487 performs the same procedure multiple times. So, the goal of this phase is to
488 provide an automated procedure valid for the DUT(s) used in the experiment.

489 In the case of instant power consumption measurement, the scripts should
490 automate both the data acquisition and the SWUT execution. When per-
491 forming a time measurement, the script must store all the system times as
492 well as manage the battery recharge to avoid human intervention. In [14] and
493 [15], the authors explained a possible implementation of this kind of scenario
494 automation for time measurements. For model measurements, the scripts
495 run all the software measurements tools defined in the previous phase and
496 collect resource usage logs for each scenario.

497 An incorrect setup of the experiment poses a threat to *Construct validity*
498 of the results since it could lead to measuring the wrong construct.

499 The output of this phase will be the scripts, which automate the data
500 collection procedure and a set of files, which contain the raw energy con-
501 sumption data according to the data format provided in the previous phase.
502 The Do Deliverable document, introduced in Table 1, will be a synthetic
503 report that lists and explains the content of each script and raw data. The
504 availability of scripts and data make the replication and verification of results
505 – essential in any scientific approach – to be carried out by third party. A
506 recommended practice, is to leverage open public repositories – e.g. figShare,
507 Zenodo, and GitHub – to store scripts and data.

508 *2.4. Phase IV: Analyse*

509 In this phase, the consumption data collected in the previous phase is
510 analysed. There are two approaches for identifying task-related data in power
511 traces:

- 512 • Online with synchronisation between the recorder and under measure-
513 ment systems, and
- 514 • Offline using added markups to the traces.

515 With the first approach, only the portion of the traces pertaining to the
516 observed tasks is recorded and later processed. The approach requires accu-
517 rate synchronisation that is based on the capability to timely communicate
518 between the device and the measurement instrumentation.

519 The second approach requires all the traces for a series of experiments to
520 be recorded, and then, during an analysis phase, the segments pertaining to
521 the observed tasks are extracted and processed. It requires no synchronisa-
522 tion as it suffices for trivial instrumentation to add markups into the traces.
523 This approach is supported by the R package Powtran². The result of the
524 power trace analysis is the total amount of energy consumed to perform a
525 task.

526 The energy consumption obtained in either way can then be analyzed
527 according to the method defined in the How phase.

528 The Analyse phase might pose a threat to the *Conclusion validity*. In
529 particular the data must be checked for the presence of outliers, which must
530 be assessed, then a decision must be taken concerning their possible removal.
531 In addition the distribution of the energy data should be identified; this is
532 important to allow the choice of the appropriate statistical tests.

533 The output of this phase is a deliverable, called the Analyse Deliverable as
534 described in Table 1, which contains Data analysis scripts, the Data analysis
535 results, and the conclusion threats to validity analysis.

536 3. Applying the Consumption Measurement Process

537 In this section, we show how the proposed process can be applied to an
538 example in which a battery-powered Raspberry Pi is used to sort integer
539 values gathered by a sensor. The experiment can be deemed as represen-
540 tative of a typical environment in which measuring the energy consumption
541 of a software application is required, since that estimation is crucial for the
542 development of embedded software [16]. In the example, it is required to
543 choose the most efficient sorting algorithm to maximize battery time. Given
544 that the issue is the battery time, all the consumption measurements will
545 be energy measurements. The following subsection is a process deliverable
546 according to our proposed framework.

²<https://github.com/SoftengPoliTo/powtran/> (Last Visited: 2019/09/22)

547 *3.1. Goal Deliverable*

548 As defined in Section 2.1 the Goal deliverable contains the research ques-
549 tions, the description of SwUT, device, context, and the external threats to
550 validity analysis.

551 *3.1.1. Research Questions*

552 **RQ1** Compare energy consumption of Counting Sort algorithm implemented
553 in C language and Merge Sort algorithm implemented in C language
554 run on Raspberry Pi version 2B in the context of Raspbian Linux OS:

555 **RQ1a** Characterise the energy consumption of the Counting Sort al-
556 gorithm implemented in C language run on Raspberry Pi version
557 2B in the context of Raspbian Linux OS;

558 **RQ1b** Characterise the energy consumption of the Merge Sort algo-
559 rithm implemented in C language run on Raspberry Pi version 2B
560 in the context of Raspbian Linux OS.

561 *3.1.2. SWUT Description*

562 The following two SWUTs are considered in the experiment:

- 563 • Counting sort: 2-pass sorting algorithm, with $O(n)$ time complexity;
- 564 • Merge sort: single-pass sorting algorithm, with $O(n \log n)$ time com-
565 plexity.

566 A brief description of the considered algorithms and the code implemen-
567 tation are reported in appendix Appendix A.1.

568 We planned five distinct dataset to test the SwUT labeled with numbers
569 from 1 to 5. The first dataset contains numbers from 0 to (DATASET_SIZE
570 -1) in ascending order and the second dataset contains numbers in descending
571 order from (DATASET_SIZE -1) to 0. The remaining three datasets contain
572 pseudo-random numbers with values between 0 and (DATASET_SIZE-1).
573 The seed is known, so the same pseudo-random numbers can be generated
574 anytime. For these data, DATASET_SIZE represents 50000 elements.

575 *3.1.3. Device Specifications and Context*

576 The most relevant hardware specifications for the tested device, a Rasp-
577 berry Pi 2B, are specified in table 4.

578 The context of the measurement is specified in table 5. The full set of
579 device specifications and the complete list of processes running during the
580 experiment is reported in appendix Appendix A.2.

Parameter	Value
CPU	900Mhz Quad-Core ARM Cortex-A7
RAM	1GB
Graphics Core	VideoCore IV

Table 4: Goal deliverable - Devices

Parameter	Value
OS	Raspbian Linux OS: Jessie Lite
Kernel Version	4.4
OS Config.	Default
No. running processes	22
Power information collection interface	ADC NI USB 6210
Power information processing	C software written with NI library ³

Table 5: Goal deliverable - Context

581 *3.1.4. Threats to External Validity Analysis*

582 The results will be valid only for Raspberry Pi version 2B, and the ex-
583 perimenters accept this restriction.

584 *3.2. How Deliverable*

585 As defined in section 2.2, the how deliverable for an Instant Power mea-
586 surement will contain the following sections: Hardware Instrumentation, Syn-
587 chronization, Sampling Frequency, File Format, Threats to Validity Analysis.

588 *3.2.1. Hardware Instrumentation*

- 589 • Voltage generator: 5V (max 2A)
- 590 • Shunt Resistor: 0,05 Ω
- 591 • ADC: National Instrument NI-6210
- 592 • Supervising device: Desktop Computer

593 *3.2.2. Sampling and Data Synchronization*

594 There will be no clocks synchronisation or post-processing data analysis.

595 *3.2.3. Sampling Frequency*

596 The selected sampling frequency is 125 khz.

597 *3.2.4. File Format*

598 The file name includes the following details about the experiment: de-
599 vice maker, device model, algorithm name, programming language, dataset
600 size, dataset label (e.g., progressive number). A sample file name can be
601 *Raspberry_2b_counting_c-5000_1*. A file content sample can be the following:

602 1,149160E+0

603 1,142452E+0

604 1,152316E+0

605 *3.2.5. Threats to Validity Analysis*

606 To limit the Threats to Internal Validity related to the correct determi-
607 nation of the consumption value for a specific process, we plan to:

- 608 • Run the experiment on a new installation of a Raspbian Lite OS to
609 minimise the number of concurrent processes,
- 610 • Measure the instant power consumption of the device in idle, and
- 611 • Subtract the idle value from the data obtained in each run (see section
612 3.2.7).

613 To limit the Threats to Construct Validity, we provide a voltage mea-
614 surement of the shunt resistor. The value logged in the file is the voltage
615 multiplied by the voltage divided by the shunt resistor value. This multipli-
616 cation will provide the instant power consumption value according to Ohm's
617 Law, $P = V I$. In this computation, we do not take into account the shunt
618 resistor temperature, which could alter our measurement. We are willing to
619 accept this error because it is not going to affect our results significantly.

620 To limit the Threats to Conclusion Validity, we will repeat each measure-
621 ment 30 times.

622 *3.2.6. Do Deliverable*

623 We will collect instant power consumption during the execution of the al-
624 gorithm. In the analysis phase, we will transform instant power consumption
625 to an energy value by computing the integral of instant power consumption
626 over the experiment time interval. For automating the experiment, we cre-
627 ated a script in the Python language, to:

- 628 • Run the data collector on the supervising device;

- 629 • Run the SwUT on the Raspberry Pi;
- 630 • Store the instant power consumption on a text file;
- 631 • Commit and push the instant power consumption file to a local git
632 repository.

633 The Raspberry Pi is connected to a router on a LAN. The supervising
634 device is also connected to the same network, so it is possible to run the
635 SwUT via SSH. The script reads the To-Do List from an input file. The To-
636 Do List includes a line representing each run of the SwUT specifying the
637 following information:

- 638 • Device Model;
- 639 • Programming Language;
- 640 • SwUT;
- 641 • Dataset Size;
- 642 • Dataset Label.

643 To repeat these operations programmatically, we created a program, called
644 executor, to run on the Raspberry Pi, which takes the following as input:

- 645 • The SwUT name;
- 646 • The Dataset size;
- 647 • The dataset Label.

648 When started, the executor will:

- 649 1. Create the dataset dynamically;
- 650 2. Run the marker;
- 651 3. Run the SwUT;
- 652 4. Repeat points 2 and 3 thirty times;
- 653 5. End.

654 The experiment automation script in python, the To-Do List file, and the
655 raw data are available online on an open repository [17].

656 Each element of the To-Do List executes the same task thirty times, as
657 described in Section 3.2.5. Each run is preceded by the implementation of
658 a marker, which allows the identification of the SwUT in the instant power
659 consumption data.

Table 6: Summary statistics of energy by algorithm

Algorithm	Mean	Median	SD	p.SW
counting sort	2.30	2.33	0.25	p < 0.001
merge sort	8.69	8.55	1.22	p < 0.001

660 *3.2.7. Analyse Deliverable*

661 Corresponding to the original RQ, we formulate the following null hy-
 662 pothesis: There is no significant difference in the central tendency of the
 663 energy consumed by the two algorithms in performing the sorting task. The
 664 significance level (α), corresponding to the risk of committing a type I error,
 665 i.e., rejecting the null hypothesis while it is true, may be assigned to the
 666 standard 5%.

667 *Analysis Results.* The distribution of the energy consumed per task can be
 668 represented graphically by means of a boxplot displayed in Figure 5

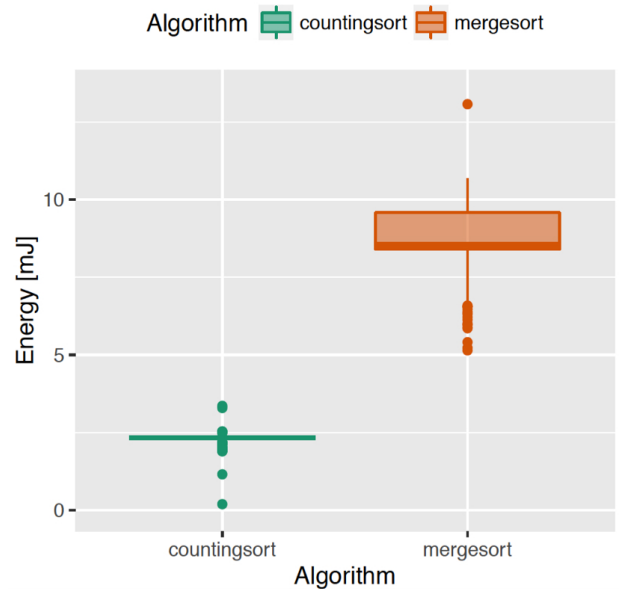


Figure 5: Energy consumed by the two algorithms for sorting an array of 50,000 elements.

669 A summary of the data, together with a central tendency, dispersion, and
 670 normality is reported in Table 6. The values are reported in millijoules.

671 We observe that the last column, reporting the p-value of the Shapiro-
672 Wilk test, contains values that are smaller than 5% (our reference value). We
673 can reject the null hypothesis for both algorithms that the values are sam-
674 pled from a normal distribution. Therefore, we should apply non-parametric
675 statistics in the following analysis. To check the null hypothesis, we can apply
676 a Mann-Whitney U test. The p-value returned by the test is smaller than the
677 reference level, so we can reject the null hypothesis. We conclude that a sig-
678 nificant difference in energy consumption exists between the two algorithms.
679 To quantify the magnitude of the difference, we compute the standardized
680 effect size. For this purpose, we adopt the Cliff's Delta statistic. We obtain
681 an effect size of -1 meaning that the amount of energy consumed by the first
682 algorithm (counting sort) is smaller than the second (merge sort) by a sig-
683 nificant amount. So, we conclude counting sort is the algorithm to select for
684 better energy efficiency.

685 4. Related Work

686 During recent years, the interest in how software influences the power
687 consumption of a device has increased sharply. It is possible to divide the
688 related work on the topic into two categories:

- 689 1. Energy consumption measurement/estimation.
- 690 2. Energy consumption reduction/optimization.

691 The first category focuses on the way in which energy is measured or es-
692 timated. A recent work by Harman et al. [7] categorizes Energy Testing
693 as one of the most important fields for Search-Based Software Engineer-
694 ing, and highlights the need for trustable metrics and for quick and well-
695 defined energy-measuring procedures. The paper also highlights several novel
696 hardware-based approaches, e.g., the SEEP [18] approach using symbolic ex-
697 ecution to capture and re-execute paths. The approach we propose is adapt-
698 able to any alternative method for measuring energy or power, since using a
699 different procedure would only have impact on the hardware section of the
700 How deliverable and on the Do deliverable where the steps of the experiments
701 are formalized.

702 Nouredine et al. [19] review different energy measurement approaches
703 that can be classified as measurement/estimation and modelling. In this first
704 sub-category, the goal is to determine the energy consumption through the
705 hardware equipment, while the latter creates a mathematical model of the

706 device energy consumption to provide energy data without external equip-
707 ment. By analysing the literature, we see that Hindle et al. [20] proposed an
708 approach to measure how the energy consumption of software applications
709 varies through the different versions. There exist working prototypes, which
710 allow estimating the energy consumption of mobile devices, the most popular
711 being DevScope [21], AppScope [22], and En-Track [23]. This work proposes
712 complete and working prototypes for measuring the power consumption of
713 Android applications. The main problem of these approaches is the limited
714 number of supported devices. For this reason, it is difficult to replicate the
715 studies to validate the measurements or to apply the same measurement on
716 a slightly different device or software.

717 The second category focuses on the changes to be made to the architecture
718 or the source code to achieve the energy consumption reduction or optimisa-
719 tion. The literature review by Aleti et al. [24] describes some approaches to
720 reduce the energy consumption by improving the software architecture.

721 All these efforts are typically individual optimisation and are difficult to
722 apply to general cases. Furthermore, both categories share some common
723 steps to be performed, such as data collection, code instrumentation, and
724 data analysis, but often it is not easy to compare the procedures in different
725 experiments since a uniform notation for the documentation of similar tasks
726 is missing. So, it is useful to think about a general process to measure
727 the power consumption of a software application, and to provide the tools
728 needed to document and analyse the data obtained in the measurement. To
729 our knowledge, such a general and repeatable approach is still missing in the
730 literature.

731 It is possible to identify many references that measure the energy con-
732 sumption of software applications and propose ways to reduce it, such as [9]
733 [25] [26] [27] [28] [29] [30] [31] [32] [33] [34] [35] [36] [37] [38] [39] [40], [41]. In
734 Table 7 we compare the information (listed as table columns) provided by our
735 process along with the information provided by each of these papers (listed
736 as table rows). A check mark (✓) indicates that the current information
737 carried out by our process is also included in the related paper. In Section
738 2 we explain in detail all information produced as the output of our process.
739 Table 7 shows that in the literature there are methods for measuring the
740 energy consumption of software applications. However, there is no common
741 procedure to extract the energy data from software applications. In detail,
742 all the analysed works lack the following features:

Table 7: Comparison between related work and our process

Related paper	[9]	[25]	[26]	[27]	[28]	[29]	[30]	[31]	[32]	[33]	[34]	[35]	[36]	[37]	[38]	[39]	[40]	[41]
Goal Deliverable																		
RQ Definition	✓		✓		✓	✓		✓	✓	✓		✓	✓		✓	✓	✓	✓
Software Under Test Description	✓			✓					✓		✓		✓	✓	✓		✓	✓
Device Context Info	✓		✓						✓		✓		✓			✓		✓
How Deliverable																		
Measurement or Estimation Technique Description	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Hardware and Software Instrumentation			✓		✓								✓			✓		✓
Sampling Frequency Used						✓	✓			✓	✓		✓	✓	✓		✓	✓
Data Format Description																		
Do Deliverable																		
Implementation Scripts Description and Publication																		
Raw Data Publication																		
Analyse Deliverable																		
Statistical Data Analysis	✓				✓	✓			✓	✓		✓	✓		✓		✓	✓
Threats to Validity Analysis	✓				✓	✓			✓	✓		✓	✓		✓	✓	✓	✓

- 743 • Provide all the information that are part of our process;
- 744 • Explain step-by-step how to replicate the experiment, and
- 745 • Provide a defined format to publish the raw data obtained.

746 Following a defined procedure will enable a comparison between data of dif-
747 ferent experiments. This will guide developers toward countermeasures to
748 handle cases of high-energy consumption. We previously identified a high-
749 level framework [42], which describes the motivations that lead to measuring
750 the energy consumption of software applications. So, the main contribution
751 of this work proposes a common process to be used by anyone to extract
752 energy data of software applications in such a way as to have comparable
753 data that is extracted and analyzed in a standard way.

754 5. Conclusion

755 The awareness of energy consumption is an emerging quality for soft-
756 ware and hardware. The expansion of mobile device usage as well as the
757 diffusion of IoT devices made energy consumption a critical issue due to
758 the limited amount of energy batteries can store. In this paper, we pre-
759 sented a well-defined and rigorous approach to plan and conduct software
760 energy consumption measurements that, to the best of our knowledge, was
761 not previously available in the literature. The proposed procedure incorpo-
762 rates features enabling the adoption of evidence-based software engineering
763 as it produces results that are:

- 764 • Trustable: detailed documentation of goals, planning, and execution
765 allows quality assessment.
- 766 • Comparable: the contextual details and the uniformity of the process
767 ease comparison.
- 768 • Actionable: the factors are defined and, thus, any energy improvement
769 actions can be properly targeted.

770 The approach is applicable in a real-world context and has been applied
771 by the authors in previous research. In addition, a sample application is
772 reported to serve as a template guide for third-party applications.

773 Furthermore, the approach also serves as a checklist for assessing existing
774 studies. We used it in this sense to evaluate the related work, as summarized
775 in Table 7.

776 For future work, we plan to create a repository where it will be possible
777 to upload the deliverables produced according to the process we describe.
778 Such repository would allow comparing different studies and building an
779 empirically backed body of knowledge.

780 **6. Statements**

781 *6.1. Conflict of Interest*

782 The authors declare that there is no conflict of interest regarding the
783 publication of this paper.

784 *6.2. Data Availability*

785 The data used to support the findings of this study are included within
786 the article in the form of references linking to resources available on the
787 figShare public open repository.

788 **References**

- 789 [1] J. Bornholt, T. Mytkowicz, K. S. McKinley, The model is not enough:
790 Understanding energy consumption in mobile devices, in: Proceedings
791 of 2012 IEEE Hot Chips 24 Symposium (HCS), pp. 1–3.
- 792 [2] P. Fairley, Blockchain world - feeding the blockchain beast if bitcoin
793 ever does go mainstream, the electricity needed to sustain it will be
794 enormous, IEEE Spectrum 54 (2017) 36–59.

- 795 [3] B. Mochocki, K. Lahiri, S. Cadambi, Power analysis of mobile 3d graph-
796 ics, in: Proceedings of the Conference on Design, Automation and Test
797 in Europe: Proceedings, DATE '06, European Design and Automation
798 Association, 3001 Leuven, Belgium, Belgium, 2006, pp. 502–507.
- 799 [4] T. Dyba, B. A. Kitchenham, M. Jorgensen, Evidence-based software
800 engineering for practitioners, *IEEE Software* 22 (2005) 58–65.
- 801 [5] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, A. Wesslén,
802 Experimentation in Software Engineering: An Introduction, Kluwer
803 Academic Publishers, Norwell, MA, USA, 2000.
- 804 [6] R. Van Solingen, V. Basili, G. Caldiera, H. D. Rombach, Goal question
805 metric (GQM) approach, *Encyclopedia of software engineering* (2002).
- 806 [7] M. Harman, Y. Jia, Y. Zhang, Achievements, open problems and chal-
807 lenges for search based software testing, in: 2015 IEEE 8th International
808 Conference on Software Testing, Verification and Validation (ICST),
809 IEEE, pp. 1–12.
- 810 [8] A. Pathak, Y. C. Hu, M. Zhang, P. Bahl, Y.-M. Wang, Fine-grained
811 power modeling for smartphones using system call tracing, in: Proceed-
812 ings of the Sixth Conference on Computer Systems, EuroSys '11, ACM,
813 New York, NY, USA, 2011, pp. 153–168.
- 814 [9] D. Di Nucci, F. Palomba, A. Prota, A. Panichella, A. Zaidman,
815 A. De Lucia, Software-based energy profiling of android apps: Sim-
816 ple, efficient and reliable?, in: 2017 IEEE 24th international conference
817 on software analysis, evolution and reengineering (SANER), IEEE, pp.
818 103–114.
- 819 [10] A. A. Nacci, F. Trovò, F. Maggi, M. Ferroni, A. Cazzola, D. Sciuto,
820 M. D. Santambrogio, Adaptive and flexible smartphone power modeling,
821 *Mobile Networks and Applications* 18 (2013) 600–609.
- 822 [11] R. Hogg, E. Tanis, *Probability and Statistical Inference*, Prentice Hall,
823 2006.
- 824 [12] R. Saborido, V. Arnaoudova, G. Beltrame, F. Khomh, G. Antoniol,
825 On the impact of sampling frequency on software energy measurements,
826 *PeerJ PrePrints* 3 (2015) e1219.

- 827 [13] B. Wheeler, M. Torchiano, *lmPerm: Permutation Tests for Linear Mod-*
828 *els*, 2016. R package version 2.1.0.
- 829 [14] L. Ardito, M. Torchiano, M. Marengo, P. Falcarin, *glcb: an energy*
830 *aware context broker*, *Sustainable Computing: Informatics and Systems*
831 *3* (2013) 18 – 26.
- 832 [15] L. Ardito, *Energy aware self-adaptation in mobile systems*, in: *Proceed-*
833 *ings - International Conference on Software Engineering*, pp. 1435–1437.
- 834 [16] M. Ibrahim, M. Rupp, H. Fahmy, *A precise high-level power consump-*
835 *tion model for embedded systems software*, *EURASIP Journal on Em-*
836 *bedded Systems 2011* (2011) 480805.
- 837 [17] R. Coppola, M. Torchiano, L. Ardito, *Methodological Guidelines for*
838 *Measuring Energy Consumption of Software Applications - Replication*
839 *Package for Raspberry PI case study*, [https://doi.org/10.6084/m9.](https://doi.org/10.6084/m9.figshare.9879503.v1)
840 [figshare.9879503.v1](https://doi.org/10.6084/m9.figshare.9879503.v1), 2019.
- 841 [18] T. Hönig, C. Eibel, R. Kapitza, W. Schröder-Preikschat, *Seep: exploit-*
842 *ing symbolic execution for energy-aware programming*, in: *Proceedings*
843 *of the 4th Workshop on Power-Aware Computing and Systems*, ACM,
844 p. 4.
- 845 [19] A. Nouredine, R. Rouvoy, L. Seinturier, *A review of energy measure-*
846 *ment approaches*, *SIGOPS Oper. Syst. Rev.* 47 (2013) 42–49.
- 847 [20] A. Hindle, A. Wilson, K. Rasmussen, E. J. Barlow, J. C. Campbell,
848 S. Romansky, *Greenminer: A hardware based mining software reposi-*
849 *tories software energy consumption framework*, in: *Proceedings of the*
850 *11th Working Conference on Mining Software Repositories, MSR 2014*,
851 ACM, New York, NY, USA, 2014, pp. 12–21.
- 852 [21] W. Jung, C. Kang, C. Yoon, D. Kim, H. Cha, *Devscope: A nonintrusive*
853 *and online power analysis tool for smartphone hardware components*, in:
854 *Proceedings of the Eighth IEEE/ACM/IFIP International Conference*
855 *on Hardware/Software Codesign and System Synthesis, CODES+ISSS*
856 *'12*, ACM, New York, NY, USA, 2012, pp. 353–362.

- 857 [22] C. Yoon, D. Kim, W. Jung, C. Kang, H. Cha, Appscope: Application
858 energy metering framework for android smartphones using kernel activ-
859 ity monitoring, in: Proceedings of the 2012 USENIX Conference on
860 Annual Technical Conference, USENIX ATC'12, USENIX Association,
861 Berkeley, CA, USA, 2012, pp. 36–36.
- 862 [23] S. Lee, W. Jung, Y. Chon, H. Cha, Entrack: A system facility for ana-
863 lyzing energy consumption of android system services, in: Proceedings
864 of the 2015 ACM International Joint Conference on Pervasive and Ubiq-
865 uitous Computing, UbiComp '15, ACM, New York, NY, USA, 2015, pp.
866 191–202.
- 867 [24] A. Aleti, B. Buhnova, L. Grunske, A. Koziolk, I. Meedeniya, Soft-
868 ware architecture optimization methods: A systematic literature review,
869 IEEE Transactions on Software Engineering 39 (2013) 658–683.
- 870 [25] C. Seo, S. Malek, N. Medvidovic, An energy consumption framework for
871 distributed java-based systems, in: Proceedings of the Twenty-second
872 IEEE/ACM International Conference on Automated Software Engineer-
873 ing, ASE '07, ACM, New York, NY, USA, 2007, pp. 421–424.
- 874 [26] C. Sahin, F. Cayci, I. L. M. Gutiérrez, J. Clause, F. Kiamilev, L. Pollock,
875 K. Winbladh, Initial explorations on design pattern energy usage, in:
876 2012 First International Workshop on Green and Sustainable Software
877 (GREENS), pp. 55–61.
- 878 [27] S. Islam, A. Noureddine, R. Bashroush, Measuring energy footprint
879 of software features, in: 2016 IEEE 24th International Conference on
880 Program Comprehension (ICPC), pp. 1–4.
- 881 [28] C. Sahin, L. Pollock, J. Clause, How do code refactorings affect energy
882 usage?, in: Proceedings of the 8th ACM/IEEE International Symposium
883 on Empirical Software Engineering and Measurement, ESEM '14, ACM,
884 New York, NY, USA, 2014, pp. 36:1–36:10.
- 885 [29] D. Li, S. Hao, W. G. J. Halfond, R. Govindan, Calculating source line
886 level energy information for android applications, in: Proceedings of
887 the 2013 International Symposium on Software Testing and Analysis,
888 ISSTA 2013, ACM, New York, NY, USA, 2013, pp. 78–89.

- 889 [30] S. Hao, D. Li, W. G. J. Halfond, R. Govindan, Estimating mobile
890 application energy consumption using program analysis, in: Proceedings
891 of the 2013 International Conference on Software Engineering, ICSE '13,
892 IEEE Press, Piscataway, NJ, USA, 2013, pp. 92–101.
- 893 [31] D. Li, W. G. J. Halfond, An investigation into energy-saving program-
894 ming practices for android smartphone app development, in: Proceed-
895 ings of the 3rd International Workshop on Green and Sustainable Soft-
896 ware, GREENS 2014, ACM, New York, NY, USA, 2014, pp. 46–53.
- 897 [32] D. Li, S. Hao, J. Gui, W. G. J. Halfond, An empirical study of the en-
898 ergy consumption of android applications, in: 2014 IEEE International
899 Conference on Software Maintenance and Evolution, pp. 121–130.
- 900 [33] D. Li, Y. Jin, C. Sahin, J. Clause, W. G. J. Halfond, Integrated energy-
901 directed test suite optimization, in: Proceedings of the 2014 Interna-
902 tional Symposium on Software Testing and Analysis, ISSTA 2014, ACM,
903 New York, NY, USA, 2014, pp. 339–350.
- 904 [34] C. Sahin, F. Cayci, J. Clause, F. Kiamilev, L. Pollock, K. Winbladh,
905 Towards power reduction through improved software design, in: 2012
906 IEEE Energytech, pp. 1–6.
- 907 [35] D. Li, Y. Jin, C. Sahin, J. Clause, W. G. J. Halfond, Integrated energy-
908 directed test suite optimization, in: Proceedings of the 2014 Interna-
909 tional Symposium on Software Testing and Analysis, ISSTA 2014, ACM,
910 New York, NY, USA, 2014, pp. 339–350.
- 911 [36] C. Sahin, P. Tornquist, R. Mckenna, Z. Pearson, J. Clause, How does
912 code obfuscation impact energy usage?, in: Proceedings of the 2014
913 IEEE International Conference on Software Maintenance and Evolution,
914 ICSME '14, IEEE Computer Society, Washington, DC, USA, 2014, pp.
915 131–140.
- 916 [37] S. Hao, D. Li, W. G. J. Halfond, R. Govindan, Estimating android
917 applications' cpu energy usage via bytecode profiling, in: Proceedings
918 of the First International Workshop on Green and Sustainable Software,
919 GREENS '12, IEEE Press, Piscataway, NJ, USA, 2012, pp. 1–7.
- 920 [38] I. Manotas, L. Pollock, J. Clause, Seeds: A software engineer's energy-
921 optimization decision support framework, in: Proceedings of the 36th

- 922 International Conference on Software Engineering, ICSE 2014, ACM,
923 New York, NY, USA, 2014, pp. 503–514.
- 924 [39] K. Liu, G. Pinto, Y. D. Liu, Data-oriented characterization of
925 application-level energy optimization, in: A. Egyed, I. Schaefer (Eds.),
926 Fundamental Approaches to Software Engineering: 18th International
927 Conference, FASE 2015, Held as Part of the European Joint Confer-
928 ences on Theory and Practice of Software, ETAPS 2015, London, UK,
929 April 11-18, 2015, Proceedings, Springer Berlin Heidelberg, Berlin, Hei-
930 delberg, 2015, pp. 316–331.
- 931 [40] K. Liu, G. Pinto, Y. D. Liu, Data-oriented characterization of
932 application-level energy optimization, in: A. Egyed, I. Schaefer (Eds.),
933 Fundamental Approaches to Software Engineering, Springer Berlin Hei-
934 delberg, Berlin, Heidelberg, 2015, pp. 316–331.
- 935 [41] D. Li, S. Hao, W. G. J. Halfond, R. Govindan, Calculating source line
936 level energy information for android applications, in: In ISSTA, pp.
937 78–89.
- 938 [42] L. Ardito, G. Procaccianti, M. Torchiano, A. Vetrò, Understanding
939 green software development: A conceptual framework, IT Professional
940 17 (2015) 44–50.
- 941 [43] R. Coppola, L. Ardito, M. Torchiano, Methodological Guidelines for
942 Measuring Energy Consumption of Software Applications - Acqui-
943 sition Software, <https://doi.org/10.6084/m9.figshare.9879569.v1>,
944 2019.

945 **Appendix A. Experiment Details**

946 *Appendix A.1. SWUT code*

947 Counting sort is a 2-pass sort algorithm that is efficient when the number
948 of distinct keys is small compared to the number of items. The first pass
949 counts the occurrences of each key in an auxiliary array, and then makes a
950 running total so each auxiliary entry is the number of preceding keys. The
951 second pass puts each item into its final place according to the auxiliary entry
952 for that key. Time complexity is $O(n)$. The implementation has been tested
953 and follows the state of the art:

```
954 void counting_sort(int A[], int n) {  
955     int i, *B,*C;  
956     B = malloc(n * sizeof(int));  
957     C = malloc(M * sizeof(int));  
958     for (i=0; i<M; i++)  
959         C[i] = 0;  
960     for (i=0; i<n; i++)  
961         C[A[i]]++;  
962     for (i=1; i<M; i++)  
963         C[i] += C[i-1];  
964     for (i=n-1; i>=0; i--) {  
965         B[C[A[i]]-1] = A[i];  
966         C[A[i]]--;  
967     }  
968     for (i=0; i<n; i++)  
969         A[i] = B[i];  
970 }
```

971 The Merge sort algorithm divides the items to be sorted into two groups,
972 recursively sorts each group, and merges them into a final, sorted sequence.
973 Time complexity is $O(n \log n)$. The implementation has been tested and fol-
974 lows the state of the art:

```
975 void my_merge_c(int *v, int dim) {  
976     int *aux;  
977     aux = (int *) malloc (dim * sizeof(int));  
978     merge_sort_recur(v,0 , dim-1, aux);  
979 }  
980
```

```

981 void merge_sort_recur(int *v, int p, int r,
982                      int *aux) {
983     int q;
984     if (p < r) {
985         q = (p+r)/2;
986         merge_sort_recur(v, p, q, aux);
987         merge_sort_recur(v, q+1, r, aux);
988         my_merge(v, p, q, r, aux);
989     }
990 }
991
992 void my_merge(int *v, int p, int q, int r,
993              int *aux) {
994     int i, j, k;
995     for ( i=p, j = q+1, k = p;
996          i<=q && j<=r; ) {
997         if (v[i] < v[j] )
998             aux[k++] = v[i++];
999         else
1000            aux[k++] = v[j++];
1001     }
1002     while ( i <= q)
1003         aux[k++] = v[i++];
1004     while ( j<=r)
1005         aux[k++] = v[j++];
1006     for(k=p; k<=r; k++)
1007         v[k] = aux[k];
1008 }

```

1009 *Appendix A.2. Devices and Context*

1010 The hardware specifications for the tested device, Raspberry Pi 2B, in-
1011 clude:

- 1012 • A 900MHz quad-core ARM Cortex-A7 CPU
- 1013 • 1GB RAM
- 1014 • USB ports: no devices connected

- 1015 • 40 GPIO pins: not used for our experiment
- 1016 • Full HDMI port: no display connected
- 1017 • Ethernet port: connected to a local router without Internet connection
- 1018 • Combined 3.5mm audio jack and composite video: not used
- 1019 • Camera interface: not used
- 1020 • Display interface: not used
- 1021 • Micro SD: Kingston 16GB Class 10
- 1022 • VideoCore IV 3D graphics core

1023 For this experiment, the context may be summarised as follows:

- 1024 • Raspbian Linux OS: Jessie Lite, Kernel version 4.4
- 1025 • Default OS configuration
- 1026 • Processes running during the experiment:
 - 1027 – kworker
 - 1028 – systemd
 - 1029 – kthreadd
 - 1030 – ksoftirqd
 - 1031 – rcu_sched
 - 1032 – rcu_bh
 - 1033 – migration
 - 1034 – kdevtmpfs
 - 1035 – netns
 - 1036 – perf
 - 1037 – khungtaskd
 - 1038 – writeback
 - 1039 – crypto

- 1040 – bioset
- 1041 – kblockd
- 1042 – rpciod
- 1043 – kswapd0
- 1044 – vmstat
- 1045 – fsnotify_mark
- 1046 – nfsiod
- 1047 – kthrotld
- 1048 – bioset

- 1049 • Power information collected through ADC NI USB 6210

- 1050 • Power information processed through custom software written in the
- 1051 C language using the default NI library. The software has been made
- 1052 available online through an open repository [43].