An Inconvenient Truth in Software Engineering? The Environmental Impact of Testing Open Source Java Projects

Andy Zaidman
a.e.zaidman@tudelft.nl
Delft University of Technology
The Netherlands
University of Victoria
Canada

ABSTRACT

As we have come to rely on software systems in our daily lives, we have a clear expectation about the reliability of these systems. To ensure this reliability, automated software quality assurance processes have become an important part of software development. However, given the climate crisis that we are witnessing, it is important to ask ourselves what the impact of all these automated quality assurance processes is in terms of electricity consumption. This study explores the electricity consumption and potential environmental impact of continuous integration and software testing in 10 open source software projects.

ACM Reference Format:
Andy Zaidman. 2024. An Inconvenient Truth in Software Engineering? The Environmental Impact of Testing Open Source Java Projects. In *5th ACM/IEEE International Conference on Automation of Software Test (AST 2024) (AST '24), April 15–16, 2024, Lisbon, Portugal.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3644032.3644461

1 INTRODUCTION

As we have grown accustomed to living in a software-filled world, we are also more and more relying on software for everyday tasks. Because of our reliance on software, its reliability is indispensable [20]. For example, it has been estimated that software failures in 2017 cost the economy $1.7 trillion [25]. Additionally, Ko et al. report on software failures that can be directly linked to the loss of 1500 human lives [24]. In this light, the role of software quality assurance becomes ever more important.

To ensure the quality of software systems, software engineers have a variety of quality assurance approaches at their disposal. Some popular approaches are: software testing [1, 4, 7, 21], modern code review [3, 5, 10, 21], automated static analysis [6, 16], and build automation [8, 11, 18, 29]. Of the four aforementioned approaches, software testing, automated static analysis, and build automation are automated and run on the workstations of software engineers, or are run through continuous integration services [8].

While we acknowledge that reliable and robust software is of the utmost importance, we cannot neglect that Information and Communication Technology (ICT) is a growing concern in the climate change debate. It has been estimated that in 2020 the energy consumption of the ICT sector reached 15% of the world's total energy consumption [15], and it has been predicted that the ICT sector could consume up to 20% of the world's electricity by 2025 [13]. How that use of electricity translates to the environmental impact is tightly related to the *carbon intensity* of the electricity. The carbon intensity expresses the "*cleanness*" of the produced electricity, i.e., it specifies how many grams of CO_2 are released to produce a kilowatt hour (kWh) of electricity [30]. The carbon intensity depends on how the electricity was produced, e.g., through renewable resources, or using fossil fuels.

Depending on the study, the overall ICT carbon footprint is broadly estimated to be between 1.8% to 3.9% [14] of the total greenhouse gas emissions as of 2020. While the impact of ICT seems modest when compared to sectors like *transportation* (27%) and the *manufacturing industry* (24%) [23], there seems to be consensus that *"urgent policy action and investment are needed to limit increases in energy use driven by increasing demand of ICT services"* [14].

Our exploratory study fits in this call to arms, as we explore — and hope to create awareness on — how popular software engineering practices are contributing to the consumption of electricity. While Pang et al. indicate that software engineers typically have little knowledge of energy consumption [28], Chowdhury et al. [9] and Verdecchia et al. [35] rightfully point out that software engineers need to have awareness about and feedback on energy consumption before they can adjust their programming practices and behaviour.

Our particular focus for this study are automations of popular software engineering practices, particularly those that we execute frequently, often without thinking about them. Two prime examples of such automations are *software testing*, and *continuous integration*. In particular, we aim to investigate how frequently they are executed and what the impact of *testing* and *performing a complete build* is in terms of electricity consumption.

Our guiding research question is the following:

> **RQ:** What is the energy impact of automated software testing and continuous integration in open source software development?

Our exploratory results indicate that there is great variety in the energy consumption among projects for these quality assurance practices. A striking example of a project that consumes quite a bit of energy is the `Elasticsearch` project: it was built 5025 times in 2022, leading to an estimated yearly energy consumption of ~161.5 kWh for building this project on an AMD Ryzen 7 CPU. This level of energy consumption corresponds to ~9.7% of the yearly average household energy consumption of a citizen in the European Union.

AST '24, April 15–16, 2024, Lisbon, Portugal

ACM ISBN 979-8-4007-0588-5/24/04.
https://doi.org/10.1145/3644032.3644461

2 STUDY SETUP

We have a very clear understanding that *we can in absolutely no way be complete in our investigation* to estimate the energy consumption of quality assurance practices in open source software (OSS). Essential reasons for this incompleteness are:

(1) We would need to have access to the precise hardware on which the build and test actions take place and be able to measure the precise power consumption.
(2) We would need to build all projects on GitHub to get a complete picture. This is infeasible from (1) a time perspective, and (2) from the perspective that it is non-trivial to get OSS projects to build out of the box. In particular, Khatami and Zaidman have shown that around 47% of the Java projects they considered for their study run out of the box, i.e., without making major changes to the configuration of their system [22]. Similar numbers have been reported by Hassan et al. (46%) [17] and Sulir et al. (41%) [31].
(3) We would need to take multiple environments into consideration, i.e., both the *workstation environment*, i.e., the hardware on which the software engineer would locally run tests, e.g., in the IDE [7] or command line, and the *continuous integration environment* [8], i.e., a server or cloud environment on which a complete build-test cycle is performed after a commit to version control.

As such, we fully acknowledge that our investigation is (1) exploratory in nature, (2) composed of a convenience sample in terms of projects, i.e., those projects that we could build locally "out of the box", i.e., we did download specific versions of the development kit, or specific compilers, but did not make changes to the source code, and (3) the energy measurements come with a number of important assumptions (see below).

2.1 Two evaluation platforms

We opted to run our energy evaluations on two separate platforms:

- A **Raspberry Pi 4 B**[1]. This mini computer is equipped with a 1.8 GHz 64-bit quad core ARM Cortex-A72 processor produced by Broadcom (model: BCM2711C0), 8GB RAM, and a Samsung 256GB mini SD card. It runs Debian GNU/Linux 11 ("Bullseye").
- A **Minisforum Mini PC** (model EM680)[2] featuring an AMD Ryzen 7 6800U 8-core processor with a base clock speed of 2.7 GHz (max. 4.7 GHz), with 16GB of RAM, and a 512GB SSD. As operating system, it runs Windows 11 Pro (version 22H2).

Our choice for these two platforms was instigated by the fact that both these devices are USB-C powered and contain no battery. As such, we could monitor their power usage with the CT-3 power meter from AVHzY[3]. We used the Shizuku Toolbox to read out electricity measurements from the CT-3 power meter[4].

This begs the question of how realistic these platforms are in terms of actual **computational performance versus electricity consumption**. On the one hand, the Raspberry Pi platform features an ARM Cortex-A72 processor design that is in use in many mid-range smart phones (e.g., Samsung's Galaxy A9). The Raspberry Pi's overall electricity footprint is less demanding with a 7.5W TDP[5]. On the other hand, the Minisforum Mini PC contains more realistic hardware with the AMD Ryzen 7 6800U chip that has an adjustable TDP of 15–28W. The chip is in use in popular notebooks such as the Asus Zenbook S 13. However, we assume build farms to contain more powerful processor designs. For example, Amazon Web Services uses Intel Xeon processors[6] that have a minimal 85W TDP[7]. We thus start from the assumption that our electricity measurements are likely at the lower end of the spectrum.

Another important factor is the **precision of the electricity consumption measurement**. We explicitly opted to measure the electricity consumption at the hardware level, and not at the software level. While measuring at the software level would be more convenient, for example, see the PeTra tool for Android electricity measurements [26], it is also less precise as it typically only considers CPU usage [19]. An alternative to a USB-C power meter would be the Monsoon power meter [2]. We avoided using a USB-C laptop, as there could be an energy draw from the battery.

2.2 Energy simulations

Because of the aforementioned assumptions that we make, and because we randomly select a commit from the year 2022 that builds successfully and that we assume to be representative for the energy consumption of all builds of a project, we refer to our study as an *energy simulation study*. We simulate two particular scenarios without Docker, as explained in our replication package [36]:

Scenario 1: Run a full build + tests. This scenario roughly corresponds to a *Continuous Integration* build. In this scenario we ensure that the project is clean and that the cache of the build system (Maven or Gradle) is empty. As such, during the build dependencies are downloaded, the entire project is built, optional analyses are executed (e.g., static analysis, or code coverage measurements), and the tests are run. We use `./gradlew build` or `mvn install`, unless otherwise specified by the documentation.

Scenario 2: Run all tests. This scenario roughly corresponds to a developer executing all the tests locally (commandline, outside of the IDE). The project is built, and we simulate the electricity consumption of running the tests. We use `./gradlew cleanTest test` or `mvn test`, unless otherwise specified by the documentation.

3 RESULTS

The results of our electricity consumption simulations can be observed in Table 1; the table also indicates the exact commit that we have considered, an estimation of the number of tests (a simple search for the occurrence of `@Test` for JUnit projects), and the number of commits on GitHub in 2022 for the particular project.

[1] See https://www.raspberrypi.com/products/raspberry-pi-4-model-b/, last visited December 1st, 2023.
[2] See https://store.minisforum.com/collections/all-product/products/minisforum-em680, last visited December 1st, 2023.
[3] See https://www.avhzy.com, last visited December 1st, 2023
[4] See https://yk-lab.org:666/shizuku/manual/software/manual-pc-en-us/content.html, last visited December 1st, 2023.
[5] TDP stands for Thermal Design Power, in watts, and refers to the power consumption under the maximum theoretical load.
[6] See: https://aws.amazon.com/intel/, last visited December 1st, 2023.
[7] https://www.intel.com/content/www/us/en/products/details/processors/xeon.html, last visited December 1st, 2023.

Table 1: Results of electricity consumption simulation using Raspberry Pi and Minisforum EM680. Single measurements are in mWh (milliwatt hour), yearly estimations are in kWh (kilowatt hour).

Project	Commit	# of tests	# of commits in 2022	Raspberry Pi				Minisforum EM680 AMD Ryzen 7 6800U			
				Energy 1 complete		Yearly energy simulation		Energy 1 complete		Yearly energy simulation	
				build	test exec.	to build	to test	build	test exec.	to build	to test
Apache Flink	f68967a	13267	3218	2325 mWh	—[a]	7.482 kWh	—	7379 mWh	3142 mWh	23.746 kWh	10.111 kWh
Apache Maven	dfbb324	1016	200	442 mWh	286 mWh	0.088 kWh	0.057 kWh	1573 mWh	918 mWh	0.315 kWh	0.184 kWh
Apache Seatunnel	bd74989	2171	1457	2763 mWh	—[b]	4.026 kWh	—	16506 mWh	—[b]	24.049 kWh	—
Cruise-control	2839592	627	68	4591 mWh	3894 mWh	0.312 kWh	0.265 kWh	8996 mWh	5254 mWh	0.612 kWh	0.357 kWh
Elasticsearch	6019f38	18275	5025	3726 mWh	2703 mWh	17.723 kWh	13.583 kWh	32131 mWh	23308 mWh	161.458 kWh	117.122 kWh
Google Guave	2786f83	3199	231	2994 mWh	2777 mWh	0.692 kWh	0.641 kWh	4969 mWh	4373 mWh	1.148 kWh	1.010 kWh
Junit5	22a64e7	1382	334	1777 mWh	466 mWh	0.594 kWh	0.156 kWh	3115 mWh	625 mWh	1.040 kWh	0.209 kWh
OpenEMS	94c7111	2738	228	2201 mWh	—[b]	0.502 kWh	—	2753 mWh	2103 mWh	0.628 kWh	0.479 kWh
Spring Boot	009d927	6946	5535	1790 mWh	—[b]	9.908 kWh	—	11727 mWh	4713 mWh	64.909 kWh	26.086 kWh
Spring Framework	2a37284	8170	2564	3066 mWh	2453 mWh	7.861 kWh	6.289 kWh	6133 mWh	3666 mWh	15.725 kWh	9.400 kWh

[a] Test execution timed out.
[b] Intermittent test failures.

Single builds. When we examine the results of individual builds, we observe a range between 442 mWh (Apache Maven) and 8050 mWh (Apache Druid) for the Raspberry Pi platform. Similarly, for the Minisforum EM680 we observe a range between 1573 mWh (Apache Maven) and 32131 mWh (Elasticsearch). The more powerful Minisforum EM680 platform uses roughly ×4 more energy for a build compared to the low-power Raspberry Pi 4 B.

Single test suite runs. Switching our attention to the energy consumption of running the test suite, we observe that for both platforms the energy consumption of a test run is typically quite a bit less than for a full build. The two extremes are still Apache Maven and Elasticsearch with an energy consumption of respectively 286 mWh and 23308 mWh. This corresponds to ∼65% and ∼73% of the energy consumption of a full build. On the faster Minisforum EM680 platform, we also observe that test runs require less energy compared to full builds, but do make the observations of the high variance between build and test energy consumption, ranging from 20% for JUnit5 to 88% for Google Guava. This is something that we aim to investigate more deeply in future work.

Turning our attention to the yearly energy consumption, an initial observation is that higher yearly energy consumption mainly stems from a higher number of commits. For example, while a single build of Apache Flink on the Minisforum EM680 platform consumes 7.379 Wh, the yearly energy consumption is 23.746 kWh due to the 3218 builds that we observed on GitHub. In contrast, we see that a project like LinkedIn Cruise-control had fewer contributions in 2022 and only required 68 builds on GitHub, leading to a yearly energy consumption for building it of 0.612 kWh (Minisforum EM680).

Initial insights with regard to Research Question. Individual runs seem to have a rather small impact in terms of electricity consumption. For example, the most energy intensive build can be observed in the Elasticsearch project (32131 mWh on the Minisforum EM680 platform). The yearly energy consumption for building Elasticsearch is ∼161 kWh, which is due to the high number of 5025 commits (and builds) that Elasticsearch underwent in 2022.

4 DISCUSSION

Electricity consumption in context. To put the energy simulation data of Table 1 into context: recharging your smartphone battery from 0 to 100% daily leads to a yearly energy consumption of ∼2 kWh[8]. At a macro level, Table 2 shows the yearly household energy consumption per citizen of a number of countries. Taking the average energy consumption of an European citizen (EU-27, the 27 countries part of the European Union), we see that this equates to 1.67 MWh. Relating this to our energy consumption simulations, we can thus see that the yearly builds of the most electricity-intensive project in our initial dataset, namely the Elasticsearch project corresponds to ∼9.7% of the average household energy consumption in the European Union (based on the number of commits in 2022 and simulated on the Minisforum EM680).

Greenhouse contribution. The *carbon intensity* is a measure of how "clean" the electricity is.[9] It is determined by the fuel mix used in the generation of the electricity.[10] As Table 2 shows, the carbon intensity, expressed in grams of CO_2 per produced kWh, varies greatly per region: 110g of CO_2 in Canada versus 531 in China. As such, it is **difficult to establish how polluting building and testing your software is**, as we need to know the carbon intensity of the electricity used to calculate the precise CO_2 emissions. If we were to assume that Elasticsearch was built in Europe, the yearly emissions would amount to 53.774 kg of CO_2.

Two platforms. We initially started our investigation with the Raspberry Pi platform, but considering the number of test runs that timed out, we switched to the Minisforum EM680 that launched in June 2023. We present data of both platforms to indicate the difference in power consumption depending on the platform. More specifically, we measured the power consumption while the computer was idle for exactly 1 hour: 1.773 Wh on the Raspberry Pi and 12.7 Wh on the Minisforum EM680.

4.1 Threats to validity

Construct validity. The documentation of the CT-3 powermeter that we use in our study reports a voltage resolution accuracy of

[8] https://www.forbes.com/sites/christopherhelman/2013/09/07/how-much-energy-does-your-iphone-and-other-devices-use-and-what-to-do-about-it/?sh=40bd24102f70, last visited December 1st, 2023
[9] https://www.nationalgrid.com/stories/energy-explained/what-is-carbon-intensity, last visited December 1st, 2023.
[10] https://shrinkthatfootprint.com/electricity-emissions-around-the-world-2/, last visited December 1st, 2023.

Table 2: Electricity consumption in the household sector per capita in a selection of countries and the associated carbon intensity of electricity production. Unless otherwise indicated, data comes from [12, 32].

Country	Household electricity consumption per capita in MWh	Carbon Intensity (grams of CO_2 per kWh)
European Union (EU-27)	1.67	334[a]
Romania	0.7	264
Italy	1.1	371
Netherlands	1.3	354
Portugal	1.4	234
Luxembourg	1.5	168
Spain	1.5	217
Germany	1.7	385
Belgium	1.7	165
Denmark	1.9	180
Sweden	4.5	45
United States	4.52 [b]	389
Canada	4.74	110[c]
China	0.43 [d]	531[e]
Brazil	0.55	140

[a] Data from 2019 [30].
[b] https://shrinkthatfootprint.com/average-household-electricity-consumption/, last visited December 1st, 2023.
[c] https://www.cer-rec.gc.ca/en/data-analysis/energy-markets/provincial-territorial-energy-profiles/provincial-territorial-energy-profiles-canada.html, last visited December 1st, 2023.
[d] https://ourworldindata.org/co2/country/china, last visited December 1st, 2023.
[e] https://www.statista.com/statistics/1300419/power-generation-emission-intensity-china/, last visited December 1st, 2023.

0.0001V. While we have not further tested this, we consider the deviation small enough to not influence our initial observations. The energy measurements might also be influenced by factors such as the network: if the network is congested, longer waiting times before dependencies are downloaded might occur and these waiting times might influence the energy consumption.

We did not run all build and test cycles multiple times due to time constraints; some single runs took a full day to complete. For Apache Maven, Apache Seatunnel, and LinkedIn Cruisecontrol we did run the full build 3 times and observed a coefficient of variance of respectively 0.009, 0.1, and 0.005, which can be characterised as *low variance*. In future work, we will solidify our findings by running the energy simulations multiple times for all projects.

External validity. The exploratory results in this paper might not be representative, as we (1) only consider a small set of software projects, and (2) simulate the energy consumption on two platforms that are known to be energy efficient. Future work needs to measure energy consumption on the actual hardware in use in data centers.

5 RELATED WORK

Verdecchia et al. take a broad look at how to make digital infrastructures more sustainable: making the software itself more energy-aware, e.g., by automatically killing zombie processes, is one of the proposed solutions [33]. Other investigations have focused on making deployed software more energy conscious, e.g., Hindle has presented the Green Mining approach to study energy consumption differences between commits [19]. Similarly, Di Nucci et al. have focused on reducing the energy consumption of Android applications [27].

In other work by Verdecchia et al., it is claimed that "*testing not only consumes most of the time and effort in a software project, but it also heavily contributes to energy waste*", without empirical foundation [34]. In contrast to the aforementioned studies, our paper contributes initial empirical insights into the electricity required to execute test suites and run continuous integration builds, in other words we focus on the energy consumption of (parts of) the software development process, not on the deployed software.

6 CONCLUSION

We have carried out an exploratory study into the energy consumption of two frequently executed software quality assurance mechanisms, namely *continuous integration* and *testing*. While we have merely simulated these two mechanisms on low-power hardware, we see indications that individual builds do not consume that much electricity. For example, 32 Wh for building Elasticsearch on the Minisforum EM680 platform with an AMD Ryzen 7 6800U CPU. Depending on the project and a myriad of factors like the size of the test suite, the elements composing the build, the number of dependencies, etc., the testing phase of the build consumes between 20% and 88% of the total energy consumption of a full build.

When considering simulated yearly totals, we do observe that somewhat larger projects like Elasticsearch that are built 5025 times yearly do consume considerable amounts of electricity: ~161 kWh for all CI builds executed in 2022. This level of energy consumption corresponds to ~9.7% of the average household energy consumption of a citizen of the European Union. Simulating the environmental cost in terms of CO_2 emissions for Elasticsearch when building it in the European Union would amount to 53.774 kg of CO_2, the equivalent of driving an average petrol car for 222 kilometers[11].

If we want to steer away from the climate crisis that we are currently experiencing, we will need to (1) further investigate the power consumption of our routine software engineering practices, (2) create awareness among software engineers of their energy impact, and (3) come up with solutions to reduce our energy footprint.

7 FUTURE PLANS

- Constructing an automated pipeline for energy measurements that would enable to simulate energy measurements at scale, for a variety of programming languages.
- Performing fine-grained measurements to isolate several steps in the build process, e.g., assembling dependencies, static analysis, compiling, testing.
- Investigating how we can reduce the number of builds by balancing (1) quick feedback to developers, and (2) energy consumption.

ACKNOWLEDGMENTS

This research was funded by the Dutch science foundation NWO through the Vici "TestShift" grant (No. VI.C.182.032).

[11] See https://www.epa.gov/energy/greenhouse-gas-equivalencies-calculator#results, last visited December 1st, 2023

REFERENCES

[1] Maurício Aniche. 2022. *Effective Software Testing: A Developer's Guide.* Manning Publications.

[2] Luca Ardito and Marco Torchiano. 2018. Creating and Evaluating a Software Power Model for Linux Single Board Computers. In *Proceedings of the 6th International Workshop on Green and Sustainable Software (GREENS).* ACM, 1—-8.

[3] Alberto Bacchelli and Christian Bird. 2013. Expectations, outcomes, and challenges of modern code review. In *35th International Conference on Software Engineering (ICSE).* IEEE Computer Society, 712–721.

[4] Gergõ Balogh, Tamás Gergely, Árpád Beszédes, and Tibor Gyimóthy. 2016. Are My Unit Tests in the Right Package?. In *2016 IEEE 16th International Working Conference on Source Code Analysis and Manipulation (SCAM).* 137–146.

[5] Moritz Beller, Alberto Bacchelli, Andy Zaidman, and Elmar Jürgens. 2014. Modern code reviews in open-source projects: which problems do they fix?. In *11th Working Conference on Mining Software Repositories (MSR).* ACM, 202–211.

[6] Moritz Beller, Radjino Bholanath, Shane McIntosh, and Andy Zaidman. 2016. Analyzing the State of Static Analysis: A Large-Scale Evaluation in Open Source Software. In *IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER).* IEEE, 470–481.

[7] Moritz Beller, Georgios Gousios, Annibale Panichella, Sebastian Proksch, Sven Amann, and Andy Zaidman. 2019. Developer Testing in the IDE: Patterns, Beliefs, and Behavior. *IEEE Trans. Software Eng.* 45, 3 (2019), 261–284.

[8] Moritz Beller, Georgios Gousios, and Andy Zaidman. 2017. Oops, my tests broke the build: an explorative analysis of Travis CI with GitHub. In *Proceedings of the International Conference on Mining Software Repositories (MSR).* IEEE, 356–367.

[9] Shaiful Chowdhury, Stephanie Borle, Stephen Romansky, and Abram Hindle. 2019. GreenScaler: training software energy models with automatic test generation. *Empirical Software Engineering* 24 (2019), 1649–1692. Issue 4.

[10] Marco di Biase, Magiel Bruntink, and Alberto Bacchelli. 2016. A Security Perspective on Code Review: The Case of Chromium. In *2016 IEEE 16th International Working Conference on Source Code Analysis and Manipulation (SCAM).* 21–30.

[11] Omar Elazhary, Colin M. Werner, Ze Shi Li, Derek Lowlind, Neil A. Ernst, and Margaret-Anne D. Storey. 2022. Uncovering the Benefits and Challenges of Continuous Integration Practices. *IEEE Trans. Software Eng.* 48, 7 (2022), 2570–2583.

[12] Eurostat. 2022. https://ec.europa.eu/eurostat/statistics-explained/index.php?title=Electricity_and_heat_statistics#Consumption_of_electricity_and_derived_heat

[13] Alcides Fonseca, Rick Kazman, and Patricia Lago. 2019. A Manifesto for Energy-Aware Software. *IEEE Software* 36, 6 (2019), 79–82.

[14] Charlotte Freitag, Mike Berners-Lee, Kelly Widdicks, Bran Knowles, Gordon S. Blair, and Adrian Friday. 2021. The real climate and transformative impact of ICT: A critique of estimates, trends, and regulations. *Patterns* 2, 9 (2021), 100340.

[15] Erol Gelenbe and Yves Caseau. 2015. The impact of information technology on energy consumption and carbon emissions. *Ubiquity* (jun 2015), 1–15.

[16] DongGyun Han, Chaiyong Ragkhitwetsagul, Jens Krinke, Matheus Paixao, and Giovanni Rosa. 2020. Does code review really remove coding convention violations?. In *2020 IEEE 20th International Working Conference on Source Code Analysis and Manipulation (SCAM).* 43–53.

[17] Foyzul Hassan, Shaikh Mostafa, Edmund S. L. Lam, and Xiaoyin Wang. 2017. Automatic Building of Java Projects in Software Repositories: A Study on Feasibility and Challenges. In *2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM).* IEEE, 38–47.

[18] Michael Hilton, Timothy Tunnell, Kai Huang, Darko Marinov, and Danny Dig. 2016. Usage, costs, and benefits of continuous integration in open-source projects. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering (ASE).* ACM, 426–437.

[19] Abram Hindle. 2015. Green mining: a methodology of relating software change and configuration to power consumption. *Empir. Softw. Eng.* 20, 2 (2015), 374–409.

[20] Mehdi Jazayeri. 2004. The Education of a Software Engineer. In *Proc. International Conference on Automated Software Engineering (ASE).* IEEE, xviii–xxvii.

[21] Yasutaka Kamei, Emad Shihab, Bram Adams, Ahmed E. Hassan, Audris Mockus, Anand Sinha, and Naoyasu Ubayashi. 2013. A Large-Scale Empirical Study of Just-in-Time Quality Assurance. *IEEE Trans. Software Eng.* 39, 6 (2013), 757–773.

[22] Ali Khatami and Andy Zaidman. 2023. State-Of-The-Practice in Quality Assurance in Java-Based Open Source Software Development. abs/2306.09665 (2023). arXiv:2306.09665 https://doi.org/10.48550/arXiv.2306.09665

[23] Keith Kirkpatrick. 2023. The Carbon Footprint of Artificial Intelligence. *Commun. ACM* 66, 8 (2023), 17–19.

[24] Amy J. Ko, Bryan Dosono, and Neeraja Duriseti. 2014. Thirty years of software problems in the news. In *Proceedings of the International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE).* ACM, 32–39.

[25] Harry McCracken. 2017. The Year That Software Bugs Ate The World. https://web.archive.org/web/20230307155438/https://www.fastcompany.com/40505226/the-year-that-software-bugs-ate-the-world

[26] Dario Di Nucci, Fabio Palomba, Antonio Prota, Annibale Panichella, Andy Zaidman, and Andrea De Lucia. 2017. PETrA: a software-based tool for estimating the energy profile of Android applications. In *Proceedings of the 39th International Conference on Software Engineering (ICSE).* IEEE Computer Society, 3–6.

[27] Dario Di Nucci, Fabio Palomba, Antonio Prota, Annibale Panichella, Andy Zaidman, and Andrea De Lucia. 2017. Software-based energy profiling of Android apps: Simple, efficient and reliable?. In *Proceedings of the International Conference on Software Analysis, Evolution and Reengineering (SANER).* IEEE, 103–114.

[28] Candy Pang, Abram Hindle, Bram Adams, and Ahmed E. Hassan. 2016. What Do Programmers Know about Software Energy Consumption? *IEEE Software* 33, 3 (2016), 83–89.

[29] Akond Rahman, Asif Partho, David Meder, and Laurie Williams. 2017. Which Factors Influence Practitioners' Usage of Build Automation Tools?. In *International Workshop on Rapid Continuous Software Engineering (RCoSE).* 20–26.

[30] Nicolae Scarlat, Matteo Prussi, and Monica Padella. 2022. Quantification of the carbon intensity of electricity produced and used in Europe. *Applied Energy* 305 (2022), 117901.

[31] Matús Sulír, Michaela Bacíková, Matej Madeja, Sergej Chodarev, and Ján Juhár. 2020. Large-Scale Dataset of Local Java Software Build Results. *Data* 5, 3 (2020), 86.

[32] Ian Tiseo. 2022. https://www.statista.com/statistics/1291750/carbon-intensity-power-sector-eu-country/

[33] Roberto Verdecchia, Patricia Lago, and Carol de Vries. 2022. The future of sustainable digital infrastructures: A landscape of solutions, adoption factors, impediments, open problems, and scenarios. *Sustainable Computing: Informatics and Systems* 35 (2022), 100767.

[34] Roberto Verdecchia, Patricia Lago, Christof Ebert, and Carol de Vries. 2021. Green IT and Green Software. *IEEE Software* 38, 6 (2021), 7–15.

[35] Roberto Verdecchia, Fabio Ricchiuti, Albert Hankel, Patricia Lago, and Giuseppe Procaccianti. 2016. Green ICT Research and Challenges. In *30th International Conference on Environmental Informatics (EnviroInfo 2016), Part I: Advances and New Trends in Environmental Informatics - Stability, Continuity, Innovation.* Springer, 37–48.

[36] Andy Zaidman. 2023. Replication package for "*An Inconvenient Truth in Software Engineering? The Environmental Impact of Testing Open Source Java Projects*". https://doi.org/10.5281/zenodo.8347120