Energy-Aware Software Testing

Roberto Verdecchia
University of Florence
roberto.verdecchia@unifi.it

Emilio Cruciani
European University of Rome
emilio.cruciani@unier.it

Antonia Bertolino
ISTI-CNR & GSSI
antonia.bertolino@isti.cnr.it

Breno Miranda
Federal University of Pernambuco
bafm@cin.ufpe.br

***Abstract*—Our planet urges for a more responsible use of its resources, and since information technology contributes substantially to the global energy consumption, software engineering research has promptly embraced this request and is actively working towards more sustainable processes. An indispensable activity in software development is testing, which is known to be very costly in terms of time and effort. On top of this, a recent study by Zaidman has shown that software testing can be a voracious energy consumer as well. In this work we introduce the very concept of energy-aware testing as the adoption of strategies designed to reduce the energy consumption of existing practices. We discuss some possible strategies and, as an example, we conduct a first study of an energy-aware variant of a simple similarity-based test prioritization approach considering both energy consumption and test suite effectiveness, which provides evidence of perceptible savings. We encourage future research in energy-aware software testing that needs to address further studies and to think up more strategies.**

I. Introduction

With the ever-growing adoption of software-intensive systems, the environmental impact of software can no longer be neglected [1]. While so far many studies in the literature focused on improving the energy efficiency of software-intensive systems [2], very little is known on the energy required to test them, yet alone on how it can be optimized. In the recent work "An Inconvenient Truth in Software Engineering? The Environmental Impact of Testing Open Source Java Projects" [3] Zaidman demonstrated that testing software systems comes at a hefty price, with the energy required to test a single software project amounting yearly to as much as 117 kWh, the same energy needed to power an average electric car for about 363 miles (584 kilometers) [4].

This issue could in the imminent future be further exacerbated by the growing adoption of Large Language Models (LLMs) for code generation tasks. As a disruptive technology, the widespread inclusion of LLM-generated code will with high probability raise the necessity of adopting more thorough and exhaustive testing practices. On top of this, with recent advancements in test automation [5] and the potential impact that LLMs will play also in test case generation, extensive test suites—which to date already count tests in the order of hundreds of millions [6]—may become even more bloated.

In this work, we focus on the question: *What can be done to optimize the energy consumption of software testing?* To be clear, ~~in this work~~ we intentionally do not focus on modifying a software under test (SUT) to improve the energy efficiency of its testing, nor do we delve into how testing practices can improve the energy efficiency of a SUT. Instead, in this paper we introduce *energy-aware software testing*, i.e., the introduction of techniques and solutions, or the adaptation of existing ones, specifically designed for the *explicit purpose of reducing the test energy consumption*. In fact, most of the solutions proposed in the literature for improving the cost effectiveness of testing, since they tend to remove redundancy and make testing more systematic, can already contribute as a by-product to reduce energy: an excellent example of such an experience can be read in the *Green Testing* callout box in [1]. Our aim here is to identify further measures that can be taken on top of already virtuous test processes.

We first explore the intersection between software testing and software environmental sustainability, by examining how different aspects of software testing, such as test prioritization, batching, and cost-aware testing can be tuned to improve the energy efficiency of testing processes (Section III). Following, we reflect upon the potential challenges that could impede the way towards making test suites more sustainable (Section IV). Finally, we make a concrete step into energy-aware software testing by reporting a small exploratory experiment (Section V), used to study the potential effectiveness of the prototypical energy-aware test prioritization algorithm GREEDY_E, designed to optimize test suite energy consumption without hindering its effectiveness. Early results show that GREEDY_E can save up to 54% energy consumption, while expectedly not impacting considerably test suite effectiveness. Closing up, we detail our future plans to refine and extensively evaluate our energy-aware test prioritization approach (Section VI), followed by a discussion of the long term follow-up research directions implied by this work.

II. Related Work

A considerable corpus of literature focuses on the environmental sustainability of software-intensive systems. Studies in this category consider heterogeneous topics [2], such as source code analyses to identify energy-greedy portions of code [7], [8], coding practices to optimize energy consumption [9], [10], energy efficiency of programming languages [11], [12], [13], and deployment strategies to limit the environmental impact of software applications [14], [15].

As potential *incipit* to the energy-aware software engineering topic, in a 2021 IEEE Software article, Verdecchia et al. stated that "*testing not only consumes most of the time and effort in a software project, but it also heavily contributes to energy waste*", without however providing any concrete evidence to support such statement [1]. In a more recent

study of 2024, Zaidman empirically showed the considerable energy consumption required to test a software-intensive system [3]. To the best of our knowledge, this is the study related the closest to our work, and the only one to date that explicitly focuses on the energy consumed *per se* by software testing. In contrast to the study of Zaidman, in this work we take a step further, by studying how software testing energy consumption can be concretely optimized, instead of exclusively measuring it. As another closely related work, Cruz and Abreu compared the energy consumption of different mobile testing frameworks, without however digging deeper into how the testing processes *per se* could be optimized other than changing testing framework [16].

Worth mentioning as related work, a set of studies focused on applying testing techniques to measure and improve the energy efficiency of a SUT, such as test case generation to assess the SUT energy efficiency [17], source code energy consumption estimation (including test code) [18], [19], and training software energy models *via* automated testing [20]. As pointed out in the introduction, while related to this work, the corpus of software testing for sustainability is deemed as out of scope for a deeper discussion in this new idea on *energy-aware software testing*, i.e., how we can improve the environmental sustainability of software testing processes themselves.

III. Envisioning the Groundwork of Energy-Aware Software Testing

Software testing is costly but necessary: because of this, in the past decades the largest part of software testing research has been devoted to reducing testing cost while maintaining as much as possible its fault-detection effectiveness. In this effort, the trend towards test automation has resulted into increasingly large test suites, which have to be maintained and executed.

Somehow the leading axiom is that the more test cases you can afford to run, the higher confidence you can gain in the proper functioning of the SUT. The forward-looking idea that we advance here is that the execution of such huge test suites can consume much energy, and the time is ripe for taking proper measures that can contain such consumption. Software testing research needs to: *i)* gain awareness of the neglected impact of software testing activities on energy consumption, and *ii)* identify strategies that can help reducing this impact (but still without impairing test effectiveness).

In the following we propose a (certainly not exhaustive) list of strategies that could be developed for this purpose.

Energy-aware test suite prioritization/reduction: There exist plenty of techniques (as surveyed, e.g., in [21], [22]) that reorder (prioritization) or select (reduction) test cases, based on some proxy criterion for their fault detection effectiveness. In both cases the aim is that of detecting possible failures after running less test cases: this aim *per se* can already contribute to reducing energy consumption. However, during the test case reordering or selection, if along with their potential effectiveness we also take into account the respective energy consumption, we could eventually obtain a more sustainable test suite. The study we present in Section V provides a first evidence in this direction. After all, more than 20 years ago Elbaum and coauthors [23] already argued that $APFD$ (Average Percentage of Faults Detected), which measures the rate of fault detection and is the *de facto* standard metric to assess test prioritization techniques, does not take into account the different costs of test cases. They thus proposed a cost-cognizant version of such metric, denoted as $APFD_c$. Along this line of reasoning, we could define a new metric $APFD_e$, where the e would be related to the respective energy consumptions of the test cases.

Follow-the-Sun test scheduling: Follow-the-Sun strategies have already been considered in software engineering [24], especially in the context of global software development projects. Here we propose to adapt this strategy, where feasible, for scheduling the execution of large test suites: the idea would be to program test execution towards regions or in clock/calendar times in which energy is green or carbon intensity is low. If we think this solution at "Google scale" [6], energy savings could be significant. Related to this strategy we can also think of adapting test batching to take into account energy consumption. In the context of continuous integration, batch testing consists of not re-executing the relevant test cases after each commit; instead multiple test cases (relative to a series of commits) are collected into groups that are then executed altogether [25]. Batching allows for saving test resources (especially when the test builds require expensive environments) and thus already contributes to reducing energy consumption. Energy-aware test batching could further enhance savings by for instance scheduling energy-heavy batches in regions where cleaner energy sources are available.

Mocking highly consuming functions: When testing large complex systems consisting of several functions, it can be the case that the large part of energy consumption is due to a limited subset of functions: for instance, functions with high-computation profiles or driving costly hardware resources. Indeed the results of our study presented later in Section V show already in Figure 1a that CPU energy consumption across the executed test cases is not uniform. Thus, to reduce energy consumption during the execution of test suites the highly consuming modules could be mocked by lightweight software emulators (we are not saying this is easy, though).

Static detection of flaky tests: Flaky tests [26], i.e., tests that exhibit unreliable behavior, are well known for their large impact on testing budget [27]: in fact, the common practice for identifying flakiness consists of repeating test execution, which of course can notably increase energy consumption. Recently, researchers have proposed strategies that can help predict the potential flakiness of test cases based on static analyses (e.g., [28], [29]). By routinely adopting one such strategy, the rerunning of test cases to identify flaky behavior could be limited only to those test cases that static analyses have pointed as potentially flaky, thus reducing energy consumption.

Predict energy consumption of tests statically: The key solution at the basis of any energy-aware testing strategy we can conceive would be the capability of predicting statically the potential level of energy consumption of a test case: with

such an estimate, in fact, we could: *i)* avoid the execution of all test cases for measuring their consumption and directly apply the above outlined strategies; and *ii)* we could possibly understand the factors that impact the most on test consumption and learn how to design test cases that consume less. There already exists a proposal to do this by applying a big-data approach [30]: in their work Chowdhury and Hindle already showed that energy consumption can be estimated based on CPU usage and system calls count. More models like theirs need to be developed for supporting energy-aware testing.

IV. Potential Challenges Ahead

Challenges related to the adoption of software sustainability practices in software engineering have at length been already discussed in the related literature [31], [32], [33], [15], [34]. In this section, we briefly comment on the impediments we subjectively deem could impact the most the domain of energy-aware software testing.

Trade-offs between test suite reliability and testing energy consumption: The first and potentially most important challenge of energy-aware software testing is intuitively the possible loss of test suite reliability in favor of energy efficiency gains. While important, active effort should be spent to balance the environmental sustainability of testing with other equally important sustainability dimensions, such as the technical, social, and economic ones. Narrowing down the larger sustainability discourse to the technical dimension, focus should be given, as already done for test suite prioritization/reduction *via* consolidated metrics and processes [21], to empirically show that energy-aware software testing techniques do not imply a significant loss in test suite reliability.

Embedding energy measurements and consumption optimization into testing processes: The effort required to modify current testing practices to include environmental sustainability considerations should not be underestimated. Despite the landscape of software energy measurement tools is making leaps in recent years, and efficient and effective tools such as CodeCarbon[1] are gaining traction, collecting sound energy measurements and integrating energy-aware testing strategies could be a substantial endeavor. Among other crucial factors, key related challenges may lie in *i)* systematically collecting accurate test energy consumption measurements, *ii)* adapting existent testing pipelines to make them energy-aware, and *iii)* ensuring the additional effort, complexity, and energy overhead introduced by energy-aware testing do not become a burden for one or more sustainability dimensions (including the environmental one itself).

Rethinking the testing landscape: From continuous integration testing to testing only when necessary, the next step envisioned in this work is to reimagine testing practices in light of the finite environmental resources they consume. This uncharted point of view might face resilience in its adoption, bring to light unknown complexities, and could potentially require considerable time to become a reality. Despite all impediments, we believe that a systematic route towards energy-aware software testing is possible.

[1]https://codecarbon.io. Accessed 10/10/2024.

V. Digging Deeper: Starting to Collect Usable Evidence *via* Test Prioritization

As hinted in Section III, test prioritization is the process of ranking and executing tests in an order that maximizes the likelihood of detecting software bugs earlier [35]. A common strategy involves similarity-based approaches [36], where tests that differ—according to some notion of dissimilarity—from those already prioritized are given higher priority, ensuring diverse software areas or functions are tested sooner [37], [38], [39]. In this section, we present a proof of concept approach that integrates energy-awareness into similarity-based prioritization, with the goal of achieving a more energy-efficient execution of the test suite. We test the approach on one software project, assessing the energy savings achieved from this prioritization strategy. While our approach is designed to balance energy optimization with test suite effectiveness, our preliminary experiments focus on investigating its potential for reducing energy consumption, leaving considerations regarding test suite effectiveness as future work (see also Section VI).

A. The Approach

Let $T = \{t_1, \ldots, t_n\}$ be a test suite, where each test t_i is represented by its source code. We embed the tests into a d-dimensional space by modeling them as vectors of features weighted by their TF-IDF score [40], with d being the number of unique tokens among all tests in T. Analogous representations solely based on the source code of the tests have already been successfully used in similarity-based software testing approaches [41], [42], [43]. We define the dissimilarity/distance of two tests as the *cosine distance* between their vector representations. Given two vectors $\mathbf{t}_1, \mathbf{t}_2 \in \mathbb{R}^d$, their cosine distance is defined as $d(\mathbf{t}_1, \mathbf{t}_2) := 1 - \frac{\langle \mathbf{t}_1, \mathbf{t}_2 \rangle}{\|\mathbf{t}_1\| \cdot \|\mathbf{t}_2\|}$, where $\langle \cdot \rangle$ denotes the dot product and $\| \cdot \|$ the Euclidean norm.

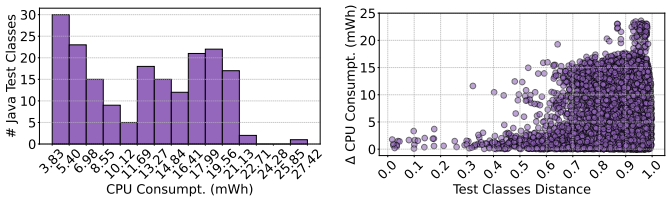
We first consider a basic similarity-based greedy approach denoted as $\textsc{Greedy}_R$, and then compare it with an energy-aware variant, $\textsc{Greedy}_E$.

$\textsc{Greedy}_R$ works as follows. Let P be the sequence of prioritized tests, which is initially empty. The first test is selected randomly and added to P. At each step, we maintain a *representative vector* $\mathbf{r} := \frac{1}{|P|} \sum_{\mathbf{t} \in P} \mathbf{t}$, i.e., the average of the vector representations of the prioritized tests. To determine the next tests to prioritize, we compute a set of *candidate tests* $C := \arg\max_{A \subseteq T \setminus P; |A| \leq \kappa} \sum_{\mathbf{a} \in A} d(\mathbf{a}, \mathbf{r})$, namely the set of κ tests that are the furthest from $\mathbf{r}$. We then pick the next $\rho \leq \kappa$ tests to prioritize uniformly at *random* from C and we add them to P. Note that ρ, κ are positive integer parameters that can be tuned to achieve different results. The algorithm proceeds until all tests are prioritized.

The energy-aware test prioritization algorithm $\textsc{Greedy}_E$ follows the same steps, except after computing the candidate set C, where the next tests to prioritize and add to P are selected as those with the *lowest energy consumption*.

B. Experimental Material

We consider Apache Maven[2] version 3.9.9 as our experimental subject. This Java project consists of 14 modules and its test suite includes 191 test classes. CPU energy consumption for each test is measured using CodeCarbon, averaged over 50 executions, on an M1 MacBook Air with 16 GB of memory.



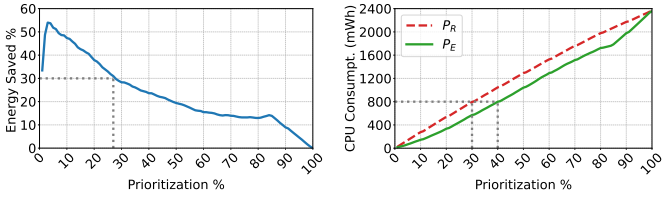
(a) Energy consumption distribution (b) Test distance and energy correlation

Fig. 1: Apache Maven test suite energy consumption data

Figure 1 provides an overview of the energy consumption of the Apache Maven test suite. Figure 1a shows the distribution of CPU energy consumption across tests, which ranges from 3.83 to 27.42 mWh, with a total consumption of around 2 400 mWh. It is clear that not all tests have the same energy cost, with some consuming up to 7 times more energy than others. Figure 1b compares the dissimilarity of test pairs with their energy consumption differences, revealing that highly similar tests tend to have comparable energy consumption, while more dissimilar tests show weaker correlation. We leverage this property with GREEDY$_E$: given the high dimensionality of the vector space, the candidate set C will very likely contain dissimilar tests with a wide range of energy consumptions, and GREEDY$_E$ will prioritize the cheap ones. In other words, diversity serves as the primary criterion for prioritization due to its proven effectiveness, while energy-data act as a secondary criterion to break ties in the prioritization, helping to reduce overall energy consumption.

C. Experimental Study

We prioritize the Maven test suite using both GREEDY$_R$ and GREEDY$_E$ and respectively generate the prioritized test suites P_R and P_E. Then for each budget $B \in \{1, 2, \ldots, 100\}$ we measure the energy consumption of running the first $B\%$ of the tests in P_R and P_E. We set the parameters $\kappa = 30$ and $\rho = 1$ for both approaches.



(a) Energy saved by GREEDY$_E$ (b) GREEDY$_R$ vs GREEDY$_E$ prioritization

Fig. 2: Experimental comparison of prioritization algorithms

Figure 2 presents the results of our experimental study. Figure 2a shows that P_E, compared to P_R, achieves an energy saving of at least 30%, with a peak of 54%, when running up to 27% of the test suite in the prioritized order. As expected, the saving decreases to 0 as the entire suite is executed, where the total energy consumption is order-independent. Figure 2b shows that, for a fixed energy budget, P_E allows to run more tests than P_R—for example, with an 800 mWh budget, P_R covers only 30% of the test suite, while P_E covers up to 40%.

Finally, the energy consumed by each prioritization algorithm is approximately 1 mWh, which is less than one third of the energy required to run the cheapest test class and is negligible compared to the cost of the entire test suite.

While the results of this study show promising savings using energy-aware test prioritization strategies, it is important to acknowledge its preliminary nature. The results herein should be viewed as insights rather than conclusions. Also note that we did not assess the effectiveness of the approach in terms of fault detection capability, but we expect it to be comparable to that of other similarity-based approaches [36].

We have made a replication package available online[3].

VI. Future Plans

To further develop the ideas proposed in this paper, we plan to work on two different temporal horizons.

Short-term horizon: To complete the preliminary research on energy-aware test prioritization presented in Section V, we plan to make the energy-aware prioritization concept more scalable and capable of handling large test suites efficiently, as other similarity-based approaches [42], [43]. In fact, while GREEDY$_E$ has a negligible computational cost for the relatively small test suite considered, this cost grows rapidly—potentially more than linearly—as test suites increase in size.

In terms of validation, as pointed out in Sections IV and V, it is crucial to assess not only the energy gains, but also the impact on test suite reliability of testing techniques. To this end, we will use software repositories with documented real-life bugs, such as Defects4J [44] and SIR [45], to assess the effectiveness of the approach in detecting bugs without resorting to fault injection. To gain more insights into the real-world applicability of our approach, we will also use open-source projects belonging to software ecosystems, e.g., Apache, and obtain fault matrices *via* mutation testing.

Long-term horizon: The research agenda implied by this work reaches far beyond the short term refinement and evaluation of energy-aware test prioritization approaches. On the one hand, in light of the challenges energy-aware software testing will face (see Section IV), we are keen to conduct *in vivo* industrial case studies to assess how energy-aware testing can be integrated in existing testing pipelines, the related effort, and challenges. On the other hand, we are eagerly and curiously driven to explore how other energy-aware testing strategies (see Section III) can be concretely implemented to study their potential in making software testing energy-aware.

[2]https://maven.apache.org. Accessed 10/10/2024.

[3]https://figshare.com/s/776b99b9328aefe1114e. Accessed 10/10/2024.

REFERENCES

[1] R. Verdecchia, P. Lago, C. Ebert, and C. De Vries, "Green IT and green software," *IEEE Software*, vol. 38, no. 6, pp. 7–15, 2021.

[2] S. Georgiou, S. Rizou, and D. Spinellis, "Software development lifecycle for energy efficiency: techniques and tools," *ACM Computing Surveys*, vol. 52, no. 4, pp. 1–33, 2019.

[3] A. Zaidman, "An Inconvenient Truth in Software Engineering? The Environmental Impact of Testing Open Source Java Projects," in *Intl. Conf. on Automation of Software Test*, pp. 214–218, 2024.

[4] L. Zhao, E. R. Ottinger, A. H. C. Yip, and J. P. Helveston, "Quantifying electric vehicle mileage in the United States," *Joule*, vol. 7, no. 11, pp. 2537–2551, 2023.

[5] J. J. Li, A. Ulrich, X. Bai, and A. Bertolino, "Advances in test automation for software with special focus on artificial intelligence and machine learning," *Software Quality Journal*, vol. 28, pp. 245–248, 2020.

[6] A. Memon, Z. Gao, B. Nguyen, S. Dhanda, E. Nickell, R. Siemborski, and J. Micco, "Taming google-scale continuous testing," in *Intl. Conf. on Soft. Eng.: Soft. Eng. in Practice*, pp. 233–242, IEEE, 2017.

[7] K. Aggarwal, A. Hindle, and E. Stroulia, "Greenadvisor: A tool for analyzing the impact of software evolution on energy consumption," in *Intl. Conf. on Software Maintenance and Evolution*, pp. 311–320, IEEE, 2015.

[8] A. Pathak, Y. C. Hu, and M. Zhang, "Where is the energy spent inside my app? fine grained energy accounting on smartphones with eprof," in *European Conf. on Computer Systems*, pp. 29–42, 2012.

[9] J. Grossschadl, S. Tillich, C. Rechberger, M. Hofmann, and M. Medwed, "Energy evaluation of software implementations of block ciphers under memory constraints," in *Design, Automation & Test in Europe Conf. & Exhibition*, pp. 1–6, IEEE, 2007.

[10] M. Linares-Vásquez, G. Bavota, C. Bernal-Cárdenas, R. Oliveto, M. Di Penta, and D. Poshyvanyk, "Mining energy-greedy api usage patterns in android apps: an empirical study," in *Conf. on Mining Software Repositories*, pp. 2–11, 2014.

[11] S. Abdulsalam, D. Lakomski, Q. Gu, T. Jin, and Z. Zong, "Program energy efficiency: The impact of language, compiler and implementation choices," in *Intl. Green Computing Conf.*, pp. 1–6, 2014.

[12] X. Chen and Z. Zong, "Android app energy efficiency: The impact of language, runtime, compiler, and implementation," in *Intl. Conf. on Big Data and Cloud Computing*, pp. 485–492, IEEE, 2016.

[13] M. Rashid, L. Ardito, and M. Torchiano, "Energy consumption analysis of algorithms implementations," in *Intl. Symp. on Empirical Soft. Eng. and Measurement*, pp. 1–4, 2015.

[14] M. Gribaudo, T. T. N. Ho, B. Pernici, and G. Serazzi, "Analysis of the influence of application deployment on energy consumption," in *Intl. Workshop on Energy Efficient Data Centers*, pp. 87–101, Springer, 2015.

[15] R. Verdecchia, P. Lago, and C. De Vries, "The future of sustainable digital infrastructures: A landscape of solutions, adoption factors, impediments, open problems, and scenarios," *Sustainable Computing: Informatics and Systems*, vol. 35, p. 100767, 2022.

[16] L. Cruz and R. Abreu, "On the energy footprint of mobile testing frameworks," *Trans. on Soft. Eng.*, vol. 47, no. 10, pp. 2260–2271, 2019.

[17] J. Mancebo, C. Calero, F. García, M. Á. Moraga, and I. G.-R. de Guzmán, "Feetings: framework for energy efficiency testing to improve environmental goal of the software," *Sustainable Computing: Informatics and Systems*, vol. 30, p. 100558, 2021.

[18] R. Pereira, T. Carção, M. Couto, J. Cunha, J. P. Fernandes, and J. Saraiva, "Helping programmers improve the energy efficiency of source code," in *Intl. Conf. on Soft. Eng. Companion*, pp. 238–240, IEEE, 2017.

[19] F. Wedyan, R. Morrison, and O. S. Abuomar, "Integration and unit testing of software energy consumption," in *Intl. Conf. on Software Defined Systems*, pp. 60–64, IEEE, 2023.

[20] S. Chowdhury, S. Borle, S. Romansky, and A. Hindle, "Greenscaler: training software energy models with automatic test generation," *Empirical Soft. Eng.*, vol. 24, pp. 1649–1692, 2019.

[21] S. Yoo and M. Harman, "Regression testing minimization, selection and prioritization: a survey," *Software Testing, Verification and Reliability*, vol. 22, no. 2, pp. 67–120, 2012.

[22] R. Greca, B. Miranda, and A. Bertolino, "State of practical applicability of regression testing research: A live systematic literature review," *ACM Computing Surveys*, vol. 55, no. 13s, pp. 1–36, 2023.

[23] S. Elbaum, A. Malishevsky, and G. Rothermel, "Incorporating varying test costs and fault severities into test case prioritization," in *Intl. Conf. on Soft. Eng.*, pp. 329–338, IEEE, 2001.

[24] J. Kroll, I. Richardson, R. Prikladnicki, and J. L. Audy, "Empirical evidence in follow the sun software development: A systematic mapping study," *Information and Software Technology*, vol. 93, pp. 30–44, 2018.

[25] E. Fallahzadeh, A. H. Bavand, and P. C. Rigby, "Accelerating continuous integration with parallel batch testing," in *European Soft. Eng. Conf. and Symp. on the Foundations of Soft. Eng.*, pp. 55–67, 2023.

[26] O. Parry, G. M. Kapfhammer, M. Hilton, and P. McMinn, "A survey of flaky tests," *Trans. on Soft. Eng. and Methodology*, vol. 31, no. 1, pp. 1–74, 2021.

[27] M. Eck, F. Palomba, M. Castelluccio, and A. Bacchelli, "Understanding flaky tests: The developer's perspective," in *European Soft. Eng. Conf. and Symp. on the Foundations of Soft. Eng.*, pp. 830–840, 2019.

[28] G. Pinto, B. Miranda, S. Dissanayake, M. d'Amorim, C. Treude, and A. Bertolino, "What is the vocabulary of flaky tests?," in *Intl. Conf. on Mining Software Repositories*, pp. 492–502, 2020.

[29] R. Verdecchia, E. Cruciani, B. Miranda, and A. Bertolino, "Know you neighbor: Fast static prediction of test flakiness," *IEEE Access*, vol. 9, pp. 76119–76134, 2021.

[30] S. A. Chowdhury and A. Hindle, "Greenoracle: Estimating software energy consumption with energy measurement corpora," in *Intl. Conf. on Mining Software Repositories*, pp. 49–60, 2016.

[31] C. Becker, R. Chitchyan, L. Duboc, S. Easterbrook, B. Penzenstadler, N. Seyff, and C. C. Venters, "Sustainability design and software: The karlskrona manifesto," in *Intl. Conf. on Soft. Eng.*, vol. 2, pp. 467–476, IEEE, 2015.

[32] P. Lago and B. Penzenstadler, "Reality check for soft. eng. for sustainability—pragmatism required," *Journal of Software: Evolution and Process*, vol. 29, no. 2, pp. 1–4, 2017.

[33] C. C. Venters, R. Capilla, S. Betz, B. Penzenstadler, T. Crick, S. Crouch, E. Y. Nakagawa, C. Becker, and C. Carrillo, "Software sustainability: Research and practice from a software architecture viewpoint," *Journal of Systems and Software*, vol. 138, pp. 174–188, 2018.

[34] C. Calero and M. Piattini, "Puzzling out software sustainability," *Sustainable Computing: Informatics and Systems*, vol. 16, pp. 117–124, 2017.

[35] G. Rothermel, R. Untch, C. Chu, and M. Harrold, "Prioritizing test cases for regression testing," *Trans. on Soft. Eng.*, vol. 27, no. 10, pp. 929–948, 2001.

[36] H. Hemmati, A. Arcuri, and L. Briand, "Empirical investigation of the effects of test suite properties on similarity-based test case selection," in *Intl. Conf. on Software Testing, Verification and Validation*, pp. 327–336, 2011.

[37] B. Jiang, Z. Zhang, W. K. Chan, and T. H. Tse, "Adaptive random test case prioritization," in *Intl. Conf. on Automated Soft. Eng.*, pp. 233–244, 2009.

[38] H. Hemmati, A. Arcuri, and L. Briand, "Achieving scalable model-based testing through test case diversity," *Trans. Softw. Eng. Methodol.*, vol. 22, Mar. 2013.

[39] R. Feldt, S. Poulding, D. Clark, and S. Yoo, "Test set diameter: Quantifying the diversity of sets of test cases," in *Intl. Conf. on Software Testing, Verification and Validation*, pp. 223–233, IEEE, 2016.

[40] J. Leskovec, A. Rajaraman, and J. D. Ullman, *Mining of massive data sets*. Cambridge university press, 2020.

[41] Y. Ledru, A. Petrenko, S. Boroday, and N. Mandran, "Prioritizing test cases with string distances," *Automated Soft. Eng.*, vol. 19, pp. 65–95, 2012.

[42] B. Miranda, E. Cruciani, R. Verdecchia, and A. Bertolino, "FAST approaches to scalable similarity-based test case prioritization," in *Intl. Conf. on Soft. Eng.*, pp. 222–232, 2018.

[43] E. Cruciani, B. Miranda, R. Verdecchia, and A. Bertolino, "Scalable approaches for test suite reduction," in *Intl. Conf. on Soft. Eng.*, pp. 419–429, AMC/IEEE, 2019.

[44] R. Just, D. Jalali, and M. D. Ernst, "Defects4j: A database of existing faults to enable controlled testing studies for java programs," in *Intl. Symp. on Software Testing and Analysis*, pp. 437–440, 2014.

[45] H. Do, S. Elbaum, and G. Rothermel, "Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact," *Empirical Soft. Eng.*, vol. 10, pp. 405–435, 2005.