Bringing Green Software to Computer Science Curriculum:

Perspectives from Researchers and Educators

João Saraiva
HASLab/INESC Tec
Universidade do Minho, Portugal
saraiva@di.uminho.pt

Ziliang Zong
Department of Computer Science,
Texas State University, USA
ziliang@txstate.edu

Rui Pereira
HASLab/INESC Tec
Portugal
rui.a.pereira@inesctec.pt

ABSTRACT

Only recently has the software engineering community started conducting research on developing energy efficient software, or green software. This is shadowed when compared to the research already produced in the computer hardware community. While research in green software is rapidly increasing, several recent studies with software engineers show that they still miss techniques, knowledge, and tools to develop greener software. Indeed, all such studies suggest that green software should be part of a modern Computer Science Curriculum.

In this paper, we present survey results from both researchers' and educators' perspective on green software education. These surveys confirm the lack of courses and educational material for teaching green software in current higher education. Additionally, we highlight three key pedagogical challenges in bringing green software to computer science curriculum and discussed existing solutions to address these key challenges. We firmly believe that "green thinking" and the broad adoption of green software in computer science curriculum can greatly benefit our environment, society, and students in an era where software is everywhere and evolves in an unprecedented speed.

CCS CONCEPTS

• **Hardware** → *Power estimation and optimization*; • **Social and professional topics** → *Software engineering education*; *Computer engineering education.*

KEYWORDS

Green Software, Green Computing, Green Education

ACM Reference Format:
João Saraiva, Ziliang Zong, and Rui Pereira. 2021. Bringing Green Software to Computer Science Curriculum:: Perspectives from Researchers and Educators. In *26th ACM Conference on Innovation and Technology in Computer Science Education V. 1 (ITiCSE 2021), June 26–July 1, 2021, Virtual Event, Germany.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3430665.3456386

ITiCSE 2021, June 26-July 1, 2021, Virtual Event, Germany.

ACM ISBN 978-1-4503-8214-4/21/06.
https://doi.org/10.1145/3430665.3456386

1 INTRODUCTION

The world is increasingly aware of and concerned about sustainability and the green movement. Computers and their software play a pivotal role in our world, thus it has a special responsibility for social development and the welfare of our planet. In this century, the situation is becoming critical since software is everywhere! The widespread use of computer devices, from regular desktop computers, to laptops, to powerful mobile phones, to consumer electronics, and to large data centers is changing the way software engineers develop software. Indeed, in the forthcoming era of AI, Internet of Things (IoT) and edge computing, there are new concerns which developers have to consider when constructing software systems. While in the previous century both computer manufacturers and software developers were mainly focused in producing very fast computer systems, now energy consumption is becoming the main bottleneck when developing such systems [37].

Despite the considerable progress in hardware energy efficiency, only recently have the programming language and software engineering communities started conducting research on developing energy efficient software, or green software. Although still in its early stage, green software research is quickly attracting more attention, as evidenced by the organization of specific research events (i.e. the ICT4S and IGSC conferences, and the GREENS, RE4SuSy, and MeGSuS workshops), and latest research publications in green data structures [19, 35, 36], green software libraries [24], green rankings of programming languages [34], green programming practices/patterns [10, 11, 23, 27, 30], green software metrics and development [6, 17], green repositories [40], energy optimization for database [25], green web [16], green cloud [15, 18, 39], green AI [7, 22, 41], and infrastructure supporting green computing research and education [44].

While research in green software is rapidly increasing, several recent studies with software engineers show that they still miss techniques and tools to develop greener software [28, 31, 37, 38]. For example, researchers surveyed programming related sub-Reddits to ask software developers on what they know regarding green software [31], and presented the following conclusion:

"The survey revealed that programmers had limited knowledge of energy efficiency, lacked knowledge of best practices to reduce software energy consumption, and were unsure how software consumes energy. These results highlight the need for training on energy consumption."

Very similar concerns were also shown by [38] where they reported that the number of questions posted on StackOverflow related to software energy consumption increased rapidly, but most of them were not answered or poorly answered. Academia is also concerned about the under-representation of sustainability and green computing in the curricula on higher education. For example,

[42] reported the findings from a targeted survey of 33 academics on the presence of green and sustainable software engineering in higher education. The major findings indicated that sustainability is under-represented in the curricula and the main reasons are: 1) lack of awareness, 2) lack of teaching material, 3) high effort required, and 4) lack of technology and tool support.

In fact, all those recent studies show that academia should not only advance state-of-the-art research in green software design, but also educate software engineers towards greener software development. Obviously, this education is best provided from the very beginning of a software engineer career. Unfortunately, today's undergraduate computer science (CS) education often fails to address our social and environmental responsibility [8]. It is our firm belief that "green thinking" and the broad adoption of green software in computer science curriculum can greatly benefit our planet and students in this rapidly evolving era of AI, super-computing, cloud computing, IoT, and edge computing.

In this paper, we conduct a study with 21 well-known researchers and educators in the green software/computing field. We present the survey results on their perspectives regarding green software education. These surveys confirm the lack of courses and educational material for teaching green software in current higher education. Moreover, we highlight three key pedagogical challenges in bringing green software to computer science curriculum and discussed existing solutions to address these key challenges.

2 PERSPECTIVES FROM RESEARCHERS AND EDUCATORS

To better understand the perspectives of researchers and educators, we invited all GREENS[1] program committee members (72 members in the past 6 years) to participate in a survey specifically designed for green software education. GREENS is a well-established ICSE (software engineering flagship conference) co-located workshop focusing on research on green software. Our survey aimed to understand three main points of view (POV):

- POV1: Those who are currently teaching green software topics thus believe they should be taught;
- POV2: Those who are *not* currently teaching green software topics but believe they should be taught;
- POV3: Those who are *not* current teaching green software topic but believe they should *not* be taught.

Figure 1 shows the conceptual flow of the survey, which contains 6 distinct phases represented as either a rectangle or a circle. The corner of each phase represents how many questions (shown as **#Q**) being asked in that phase. Two of the phases have only one question, which separate the survey into the 3 aforementioned POVs. POV1 path presents a total of 12 questions, POV2 presents 9 questions, and POV3 presents 4 questions.

2.1 The Survey

We received 21 anonymous responses (29.2% of the PC members). The detailed results are presented in this section where **Q** indicates survey questions and **R** indicates survey results.

[1]Most recent GREENS workshop: http://greens.cs.vu.nl/

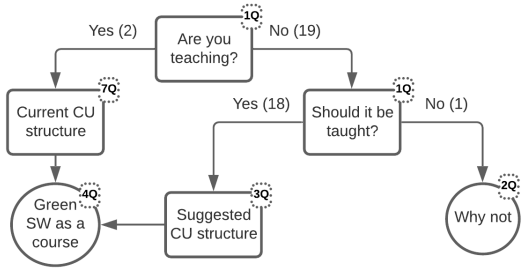

Figure 1: Survey conceptual flow

Are you teaching? Out of the 21 responses, 90.5% (19) said they were not currently teaching a Curricular Unit (CU) on green software/computing and 9.5% (2) said they were.

Should it be taught? When asking the 19 participants if they *"believe software engineering students should be taught energy efficient practices"*, 94.7% (18) said Yes, while only 5.3% (1) said No.

Why not? The single participant who believed students should not be taught energy efficient practices was asked:

(1) ***Q****: "On a scale of (Highly Disagree) 1-5 (Highly Agree), do you agree that students already have the necessary knowledge to practice energy efficient software development?"*
R: Their answer was a middle-grounded 3/5.
(2) ***Q****:"Could you briefly explain why you consider that it should not be taught?"*
R: To summarize, the participant believes that *"As someone, who is now developing in industry, I do not see it realistic to add even more requirements to developers that are already working with very many perspectives on software."*

Suggested CU structure. The answers to the following questions are visually represented in Figure 2.

(1) ***Q****: "On a scale of (Very Low Importance) 1-5 (Very High Importance), how important is teaching energy efficiency in a software engineering/IT degree?"*
R: The majority of participants ranked the importance of teaching energy efficiency in a SWE/IT degree as either high (56%) or very high (28%).
(2) ***Q****:"How should such concepts be taught?"*
R: A large majority (72%) believe that these concepts should be taught as part of already existing CUs, with 16% considering it should be a CU by itself, and the remainder 11% stating it should be taught across multiple CUs.
(3) ***Q****:"In which degree level should it be taught?"*
R: This question allowed participants to choose more than one degree levels. In this case, 77% believed it should be taught at the Master's degree level and 17% thought it should be taught at the Ph.D. level, while 61% believed it should also be included in the undergraduate curriculum.

Current CU structure. This phase was presented to two participants who are already teaching a CU related to energy efficiency/sustainability/green computing.

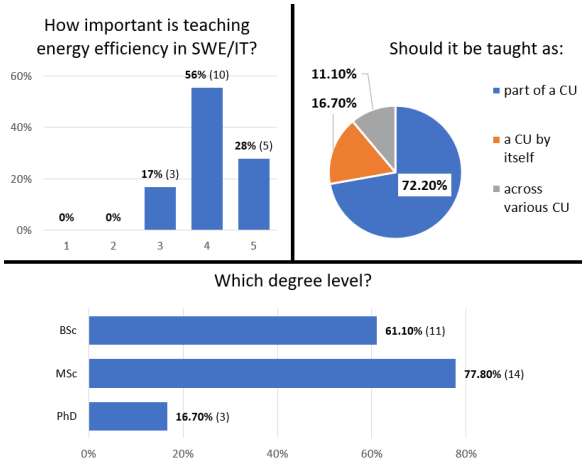

Figure 2: Results for Suggested CU structure

(1) ***Q****: "Are you teaching energy efficiency practices as a standalone CU, or within another CU?"*
R: Both participants stated that they currently teach their CU by itself.
(2) ***Q****:"What is the name of the CU?"*
R: One of the participants indicated that they teach two distinct CUs: Sustainable SW Technology (MSc) and Environmental and Sustainability Informatics (BSc). The second participant's CU is named SW Engineering and Green IT.
(3) ***Q****:"In which degree level is it taught?"*
R: Both participants answered that they teach their CU at a MSc level, while one also teaches such topics at a BSc level.
(4) ***Q****:"How many years have you been teaching this CU?"*
R: Both participants have taught their CU over 4+ years.
(5) ***Q****:"How many students enroll on average?"*
R: The number of students on average varied between 30-40.
(6) ***Q****:"What is your opinion on how the CU is performing?"*
R: The participants mentioned very positive comments regarding their CUs. They state that *"students are very interested"* and *"there is great engagement of students in the practical project, where they design and perform empirical experiments on mobile apps"*.
(7) ***Q****:"Would you like to share with us the web-page/material of your curricular unit?"*
R: Both sent their teaching materials to us but they also requested to not share this information publicly.

Green Software as a course? This final phase concludes our survey for those who both already teach a CU on sustainable IT/-green software, or believe it should be taught. In total, this phase contained 20 participant responses. The last 3 questions which are presented here not only had a list of options for the participants to choose from, but also allowed them to add any non-existing option if they believed something was missing. The answers to the following questions are visually represented in Figure 3.

(1) ***Q****: "On a scale of (Highly Disagree) 1-5 (Highly Agree), do you agree that there is enough teaching material available to support a course on green software/computing?"*
R: The majority disagrees with this comment, with 25% highly disagreeing and 40% disagreeing. This is very much in line with previous studies which have shown that there is indeed a heavy lack of knowledge and tools for green software and sustainable IT development [28, 31, 37, 38].

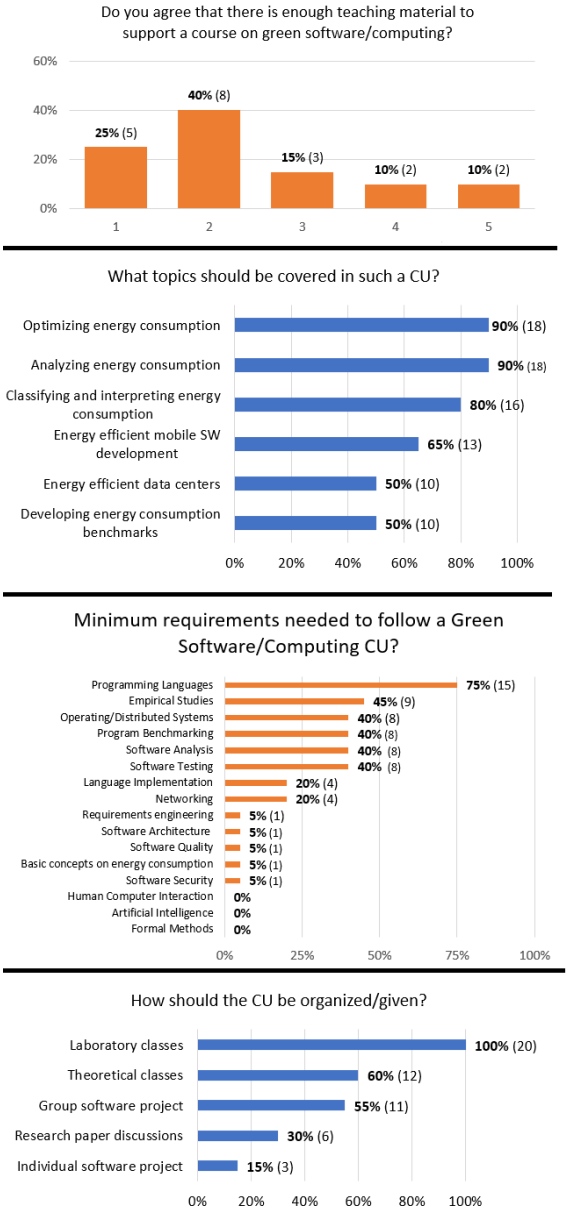

Figure 3: Results for Green SW as a course

(2) ***Q****:"What topics should be covered in such a CU?"*
R: Most participants (90%) believe that topics on how to analyze and optimize the energy consumption of software should be covered. The classification and interpretation of energy consumption results is also very much agreed upon

by many (80%), which in some sense depends on the analysis/optimization to already be present and taught. The remainder, which are more specialized topics (i.e. mobile or data centers), split the opinions of our participants in half.

(3) ***Q**:"What minimum requirements do you believe are needed for a computer science student to properly follow such a CU?"*
R: The results are overwhelmingly (75%) in favor of programming languages being the most needed minimum requirement for such a CU. Afterwards, most find knowledge on empirical studies (45%) slightly ahead of topics such as operating/distributed systems, benchmarking, and software analysis and testing (all 40%).

(4) ***Q**:"How should such a CU be organized/given?"*
R: The results show how all (100%) participants believe, without a doubt, that such a CU should be given as laboratory classes. A hands on approach to such topics are seen as the best methodology for teaching. Slightly over half (60%) believe theoretical classes should be involved in someway, and 55% believe a group software project should take place.

This survey confirms that green software/sustainable IT researchers/educators do share the concerns shown in recent studies with software engineers [28, 31, 37, 38]. The large majority believe that green software should be taught as part of already existing CUs both at the Bachelor and the Master levels. While our participants highly agree that such a course should be taught in more practical hands-on classes and labs, they do agree that the teaching materials and lab supporting tools are currently inadequate.

3 KEY PEDAGOGICAL CHALLENGES

The quickly surging demand for energy efficient computing makes it no longer sufficient for traditional computer science curriculum to train our students with only performance-oriented programming skills and mindset. It is paramount to encourage students to "think green" and write greener code. Regrettably, green software is underrepresented in current CS curricula of higher education. According to the literature [28, 31, 37, 38, 42] and our survey results presented in Section 2, a number of key pedagogical challenges prevent green software from being broadly adopted by the CS curricula. In this section, we highlight these key pedagogical challenges and present some existing solutions and guidelines.

3.1 Organization, Objectives, and Covered Topics of Independent Courses

To create an independent new green computing or green software course, the first pedagogical challenge educators must face is what should be the appropriate course objectives, organization, and covered topics? Here we present the sample curricula of two independently taught green software/computing courses (one at the Master level and the other at the Ph.D. level).

3.1.1 Master Level Green Software Engineering Course.

Course Organization: This course is offered as a one semester long elective course as part of the master program on software engineering at the University of Minho. Students are required to have strong background on programming languages. They will spend 3 hours per week in the classroom, which include a one hour seminar where all theories and techniques are presented and two hours lab where students apply the learned theories and techniques to improve software energy efficiency.

Course Objectives:

- Be able to instrument, monitor and measure the energy consumption of software systems.
- Become aware of the impact of programming practices on energy consumption.
- Become familiar with the research problems in the field of green software engineering.

Covered topics: The course covers both traditional software analysis topics, and software testing topics. For each topic, traditional state-of-the-art techniques and tools are presented. Each topic is also presented with a *green flavor*. For example, strategic [43] and aspect oriented programming [21] are both presented to compute metrics and transform/refactor source code, but also to instrument the source code without adding the intrusive energy monitoring code, keeping it in one aspect, later weaving to the program under (energy) analysis. To monitor energy consumption, we teach students how to use the Running Average Power Limit (RAPL) framework (developed by Intel [12]), where they instrument the program's source code with calls to RAPL in the lab classes. We also present a catalog of *red* smells (as described in the green software literature) and their corresponding *green* refactorings. For example, we present the *greeness* of Java collections as reported in [19, 35], and students can use the jStanley tool [36] to automatically refactor collections for energy efficiency. Also, the concepts of energy debt [9], which mimics the technical debt metaphor, and the E-Debitum plugin for SonarQube [26] are introduced, in order to compute the energy debt of Java programs. As abnormal energy consumption can be seen as software faults, we teach a variant of fault localization techniques (e.g. Spectrum-based Energy Leak Localization (SPELL) [32, 33]) to locate energy leaks in the source code. Students can use the SPELL tool to locate such energy hotspots in their software. Finally, combining SPELL with our green refactoring, we introduce automated energy-aware program repair.

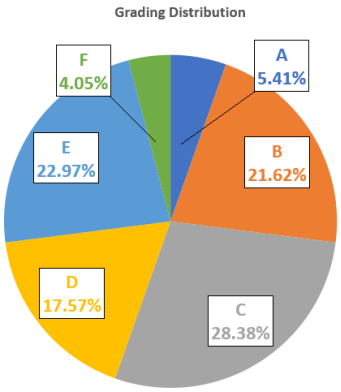

Figure 4: Green Software exam question grade distribution

During the last 3 years of the course exam, partial source code of a Java class containing 6 occurrences of different red smells were

presented to students, where they had to identify and refactor so as to improve its energy consumption. Figure 4 shows the individual European grades that students got throughout those years. These results show that students, after being trained in green software approaches, are able to directly apply them in practice. More specifically, 27% of students (received *A* or *B*) can identify and refactor at least 5 out of the 6 energy smells and the other 45% of students (with *C* or *D*) can find and fix 3 or 4 energy smells. Students who received lower grade (*E* or *F*) simply did not put in enough efforts because they also fail the full exam with very few exceptions.

3.1.2 Ph.D. Level Green Computing Course.

Course Organization: This course is offered as a one semester long elective course as part of the Ph.D. program in Computer Science at the Texas State University. It comprises of lectures, research paper reading and writing, as well as hands-on experiences on green software design. The instructor uses the state-of-the-art research papers in green computing as the textbook. Students are required to select 14 research papers from the instructor provided list, read one research paper, and submit the summary about that paper as homework each week. In addition, every student needs to choose 2-5 research papers (depending on class enrollment) to present in class and lead class discussions of their presented papers under the guidance of the instructor. Students also need to conduct a semester long research project (in group or individually) on a topic related to green computing. Students are highly encouraged to propose their own research ideas but need consent from the instructor to ensure its suitability. Students will use instructor provided research ideas in case they cannot come up with their own. The final deliverable for the research project could be a research paper, a technical report, or a software toolkit that has noticeable contribution, plus a final presentation at the end of the semester. To allow students obtain hands-on experiences on green software design and apply what they have learned from class discussions and research papers, the instructor hosts a green programming competition towards the end of the course. The difficulty of the competition problem is at the medium level of the ACM International Collegiate Programming Contest (ICPC) [3]. Students can choose different programming languages, algorithms, data structures, CPU/GPU with the goal to minimize the energy consumption when solving the same problem with identical input size. For fair comparison, all codes are submitted to and evaluated by the Greencode cloud programming portal [1, 2]. All students are required to present their energy optimization techniques (to improve student engagement and peer learning) and the instructor announces the winner after students' presentations.

Course Objectives:

- Be able to understand the state-of-the-art research and best practices of industry in green computing.
- Be able to identify and address new research problems in green computing.
- Be able to understand the impact of different programming practices on software energy efficiency.

Covered topics: This course covers a broad category of hardware and software techniques to improve the energy-efficiency of computing systems. Topics include history and road map of hardware energy efficiency, best practices in building green data centers, theories and practices of reducing software energy consumption, energy-aware resource management and scheduling, hands-on experiences on power-measurable systems and software optimizations for energy efficiency. In addition, two weeks of class time are reserved to cover emerging research topics such as green AI, power-aware 5G, and edge computing.

3.2 Integrating Green Modules to Courses

Creating an independent green computing/software course is often too time consuming or impractical due to the lack of resources and expertise. This leads to the second pedagogical challenge, which is how to integrate green modules to existing courses with a much lower cost. Introducing the concept of *green thinking* "early" and "often" to CS curriculum can effectively address this challenge. "Early" means to demonstrate the impact of programming practices on software energy consumption at the CS1 or CS2 programming courses level. "Often" means to keep refreshing and enhancing students' *green thinking* mindset across upper level undergraduate core courses such as data structures, algorithms, programming languages, software engineering, and operating systems, even to graduate level courses such as parallel and distributed computing and machine learning. For example, the following "Prime and Happy Number" project is suitable for CS1 or CS2 courses.

Project Description: Write a program that calculates how many Happy Prime numbers (numbers that are both prime and happy) between 1 and 1 million. A happy or sad number is defined by the following process: Starting with any positive integer, replace the number by the sum of the squares of its digits, and repeat the process until the number either equals 1 (happy) or loops endlessly (sad).

Project timeline: This project is assigned in two phases. In the first phase, students submit a program that calculates the correct result (There are 11,144 Happy Prime numbers between 1 and 1 million) within certain amount of time (e.g. < 60s). After the first phase, students are taught how to reduce the energy consumption of programs (not using the Happy Prime example). In the second phase, students are asked to optimize their code submitted in phase 1 for better energy efficiency and the winner will receive extra credits. To ensure fair comparison, all codes need to be submitted via GreenCode [1, 2], which can measure the energy consumption of a submitted program. Students can choose different programming languages (GreenCode supports about 20 languages), data structures, and algorithms with the goal to minimize the energy consumption of their Happy Prime code.

Figure 5 shows the energy consumption results of students' first and second submissions. Before training, students submitted programs that are highly energy inefficient. For example, 15% of students submitted codes that consumed over 1,000J of energy and no one submitted a solution that consumed less than 5J. After training, 23% of students submitted codes that consumed less than 5J and no one submitted code that consumed over 1,000J. This study clearly demonstrates the great improvement in energy efficiency that can be achieved when students are trained and motivated in writing greener code. More importantly, students feel that they have great achievements and lots of fun in participating in the competition.

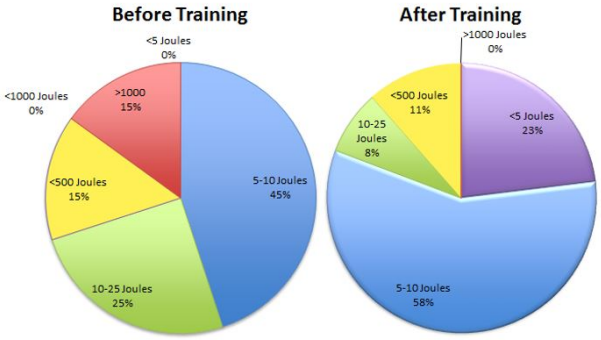

Figure 5: HappyPrime Energy Consumption of Students' Submissions - Before Training and After Training

A series of such exemplary projects specifically designed for different teaching topics can greatly help instructors integrate green modules early and often at various levels of CS courses. For example, similar projects or in-class code demos can be designed to illustrate the impact of different data structures, algorithms, programming languages, and design patterns on energy consumption and easily integrated to corresponding core courses such as data structure, algorithm design and analysis, programming languages, and software engineering. We strongly recommend integrating green computing/software modules "early" and "often" to existing courses in a way that enhances what is already taught and that melds naturally in a given course.

3.3 Lab Supporting Infrastructure

It is hard to train students how to write green software if we cannot measure the energy consumption of software. Therefore, easy-to-use tools that can provide detailed and accurate power measurement play a critical role in green computing and green software education. The lacking of such tools and low cost infrastructure that can support lab experiments in green software courses is clearly one of the key obstacles that prevent educators from introducing green computing/software topics to the CS curriculum.

Fortunately, as power consumption becomes increasingly important, most vendors today provide power measurement APIs such as Intel's Running Average Power Limit (RAPL) [12, 14] and Nvidia's Management Library (NVML) [5]. These APIs provide capabilities to measure the real-time power consumption of CPU, DRAM, and GPU for programs running on desktop workstations or servers. Since power data is accessed through machine specific registers (MSRs), users may need special permission to log in an Linux OS to obtain such data, which is not trivial for some students (especially undergraduate students). The GreenCode cloud programming portal [1, 2] provides an effective solution by allowing users to submit code from anywhere at anytime via a web browser, evaluate their energy efficiency, and share their energy efficient programs with the community for free. Since 2015, GreenCode has received nearly 30,000 code submissions and served users from the United States, Asia, Europe, Canada, and South America. It provides instructors and students a free "virtual lab" in the cloud, which significantly reduces the time and cost in setting up a similar power measurable system by themselves. GreenSoft [4] is another platform that allows researchers, educators, and students to publish blogs and share their latest research, teaching and learning experiences in green computing. It helps foster and grow the research and education community in green computing and green software design.

A number of useful tools are also available for energy analysis of applications running on mobile devices. For example, the Android Qualcomm Trepn app [20] can profile hardware usage (GPS, Wifi, etc.), resource usage (memory, CPU), and the power consumption of both the system and any standalone Android app that run on a Snapdragon chipset. PETRA [13] is another model-based tool that can estimate the energy consumption of Android apps at a coarse-grained level. Hardware-based power measurement tools that provide high frequency and high precision profiling (e.g. Monsoon [29]) are also available but they are generally expensive.

To better support the integration of green modules in CS curriculum, more easy-to-use power measurement tools and lab supporting infrastructure will be essential. It is beneficial and urgent to have a long-term plan to fund and support the development of such infrastructure/services and make them freely available to researchers, educators, and students in the green computing community.

4 CONCLUSION AND ENGAGEMENT

To conclude, we firmly believe that "green thinking" and the broad adoption of green software in computer science curriculum can greatly benefit our planet, our society, and our students in this rapidly evolving era of AI, super-computing, cloud computing, big data, IoT, and edge computing. Unfortunately, today's computer science curriculum lacks teaching materials, creative thinking, and innovative education in green computing and green software design. In this paper, we present survey results from the researchers' and educators' perspective on green software education, highlight the key pedagogical issues in teaching green software, and provide exemplary solutions to address these key challenges.

Nevertheless, bringing green software to CS core curriculum requires a long-term community effort, which includes researchers, industry pioneers, and educators working together, and periodically providing guidance on restructuring standard curricula across various courses to incorporate state-of-the-art knowledge and the best practices of green software design. We sincerely welcome all stakeholders from industry and academia to join us in making green software more attractive to students at various levels, and accelerating the formation of a community that values software energy efficiency and promotes green software design.

ACKNOWLEDGMENTS

This work is supported by the national funds through the Portuguese Funding Agency (FCT - Fundação para a Ciência e a Tecnologia, within project UIDB/50014/2020) and the U.S. National Science Foundation (NSF) under grant no. CNS-1305359. We also thank the reviewers for their valuable comments and acknowledge the support of the Erasmus+ Key Action 2 project No. 2020-1-PT01-KA203-078646: "SusTrainable - Promoting Sustainability as a Fundamental Driver in Software Development Training and Education".

REFERENCES

[1] 2018. GreenCode. https://greencode.cs.txstate.edu/.

[2] 2018. How to use GreenCode. https://greensoft.cs.txstate.edu/index.php/2018/05/27/how-to-use-the-greencode-website/.

[3] 2020. The ACM International Collegiate Programming Contest . https://icpc.global/.

[4] 2021. GreenSoft: A Community Supporting Research and Education in Energy Efficient Software Design. https://greensoft.cs.txstate.edu.

[5] 2021. NVIDIA Management Library. https://developer.nvidia.com/nvidia-management-library-nvml.

[6] Sarah Abdulsalam, Ziliang Zong, Qijun Gu, and Meikang Qiu. 2015. Using the Greenup, Powerup, and Speedup metrics to evaluate software energy efficiency. In *Proceedings of the 6th International Green and Sustainable Computing Conference*. IEEE, 1–8.

[7] Cody Blakeney, , Xiaomin Li, Yan Yan, and Ziliang Zong. 2021. Parallel Blockwise Knowledge Distillation for Deep Neural Network Compression. *IEEE Transactions on Parallel and Distributed Systems* 32 (2021), 1765–1776.

[8] Yu Cai. 2010. Integrating Sustainability into Undergraduate Computing Education. In *Proceedings of the 41st ACM Technical Symposium on Computer Science Education* (Milwaukee, Wisconsin, USA) *(SIGCSE '10)*. Association for Computing Machinery, New York, NY, USA, 524–528. https://doi.org/10.1145/1734263.1734439

[9] Marco Couto, Daniel Maia, João Saraiva, and Rui Pereira. 2020. On Energy Debt: Managing Consumption on Evolving Software. In *Proceedings of the 3rd International Conference on Technical Debt* (Seoul, Republic of Korea) *(TechDebt '20)*. Association for Computing Machinery, New York, NY, USA, 62–66. https://doi.org/10.1145/3387906.3388628

[10] Marco Couto, João Saraiva, and João Paulo Fernandes. 2020. Energy Refactorings for Android in the Large and in the Wild. In *2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 217–228.

[11] Luis Cruz and Rui Abreu. 2017. Performance-based Guidelines for Energy Efficient Mobile Applications. In *Proceedings of the 4th International Conference on Mobile Software Engineering and Systems* (Buenos Aires, Argentina) *(MOBILESoft '17)*. IEEE Press, Piscataway, NJ, USA, 46–57. https://doi.org/10.1109/MOBILESoft.2017.19

[12] Howard David, Eugene Gorbatov, Ulf R Hanebutte, Rahul Khanna, and Christian Le. 2010. RAPL: memory power estimation and capping. In *International Symposium on Low-Power Electronics and Design (ISLPED), 2010 ACM/IEEE*. IEEE, 189–194.

[13] 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 *2017 IEEE 24th international conference on software analysis, evolution and reengineering (SANER)*. IEEE, 103–114.

[14] Martin Dimitrov, Carl Strickland, Seung-Woo Kim, Karthik Kumar, and Kshitij Doshi. 2015. Intel® Power Governor. https://software.intel.com/en-us/articles/intel-power-governor. Accessed: 2017-10-12.

[15] Bradford Everman, Narmadha Rajendrana, Xiaomin Li, and Ziliang Zong. 2021. Improving the Cost Efficiency of Large-scale Cloud Systems Running Hybrid Workloads - A Case Study of Alibaba Cluster Traces. *Journal of Sustainable Computing* (2021).

[16] Bradford Everman and Ziliang Zong. 2018. GreenWeb:Hosting High-Load Websites Using Low-Power Servers. In *2018 International Green and Sustainable Computing Conference (IGSC'18)*.

[17] Blake Ford and Ziliang Zong. 2021. PortAuthority: Integrating Energy Efficiency Analysis into Cross-Platform Development Cycles via Dynamic Program Analysis. *Journal of Sustainable Computing* (2021).

[18] Keke Gai, Meikang Qiu, Hui Zhao, Lixin Tao, and Ziliang Zong. 2016. Dynamic energy-aware cloudlet-based mobile cloud computing model for green computing. *Journal of Network and Computer Applications* 59 (2016), 46–54.

[19] Samir Hasan, Zachary King, Munawar Hafiz, Mohammed Sayagh, Bram Adams, and Abram Hindle. 2016. Energy profiles of java collections classes. In *Proceedings of the 38th International Conference on Software Engineering*. ACM, 225–236.

[20] Mohammad Ashraful Hoque, Matti Siekkinen, Kashif Nizam Khan, Yu Xiao, and Sasu Tarkoma. 2015. Modeling, Profiling, and Debugging the Energy Consumption of Mobile Devices. *ACM Comput. Surv.* 48, 3, Article 39 (Dec. 2015), 40 pages. https://doi.org/10.1145/2840723

[21] Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Lopes, Jean-Marc Loingtier, and John Irwin. 1997. Aspect-oriented programming. In *ECOOP'97 — Object-Oriented Programming*, Mehmet Akşit and Satoshi Matsuoka (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 220–242.

[22] Da Li, Xinbo Chen, Becchi Michela, and Ziliang Zong. 2016. Evaluating the Energy Efficiency of Deep Convolutional Neural Networks on CPUs and GPUs. In *2016 IEEE International Conference on Sustainable Computing and Communications*.

[23] Ding Li and William G. J. Halfond. 2014. An Investigation into Energy-saving Programming Practices for Android Smartphone App Development. In *Proceedings of the 3rd International Workshop on Green and Sustainable Software* (Hyderabad, India) *(GREENS 2014)*. ACM, New York, NY, USA, 46–53. https://doi.org/10.1145/2593743.2593750

[24] Mario Linares-Vásquez, Gabriele Bavota, Carlos Bernal-Cárdenas, Rocco Oliveto, Massimiliano Di Penta, and Denys Poshyvanyk. 2014. Mining energy-greedy API usage patterns in Android apps: an empirical study. In *Proceedings of the 11th Working Conference on Mining Software Repositories*. ACM, 2–11.

[25] Divya Mahajan, Cody Blakeney, and Ziliang Zong. 2019. Improving the Energy Efficiency of Relational and NoSQL Databases via Query Optimizations. *Journal of Sustainable Computing* 22 (2019), 120–133.

[26] D. Maia, M. Couto, J. Saraiva, and R. Pereira. 2020. E-Debitum: Managing Software Energy Debt. In *2020 35th IEEE/ACM International Conference on Automated Software Engineering Workshops (ASEW)*. 170–177. https://doi.org/10.1145/3417113.3422999

[27] Sepideh Maleki, Cuijiao Fu, Arun Banotra, and Ziliang Zong. 2017. Understanding the Impact of Object Oriented Programming and Design Patterns on Energy Efficiency. In *2017 International Workshop on Sustainability in Multi-Many-Core Systems in conjunction with International Green and Sustainable Computing Conference*. IEEE.

[28] Irene Manotas, Christian Bird, Rui Zhang, David Shepherd, Ciera Jaspan, Caitlin Sadowski, Lori Pollock, and James Clause. 2016. An Empirical Study of Practitioners' Perspectives on Green Software Engineering. In *Proceedings of the 38th International Conference on Software Engineering* (Austin, Texas) *(ICSE '16)*. Association for Computing Machinery, New York, NY, USA, 237–248. https://doi.org/10.1145/2884781.2884810

[29] Monsoon. 2018. Monsoon Solutions, Inc. http://www.msoon.com/LabEquipment/PowerMonitor/.

[30] R. Morales, R. Saborido, F. Khomh, F. Chicano, and G. Antoniol. 2018. EARMO: An Energy-Aware Refactoring Approach for Mobile Apps. *IEEE Transactions on Software Engineering* 44, 12 (2018), 1176–1206.

[31] C. Pang, A. Hindle, B. Adams, and A. E. Hassan. 2016. What Do Programmers Know about Software Energy Consumption? *IEEE Software* 33, 3 (2016), 83–89.

[32] Rui Pereira, Tiago Carção, Marco Couto, Jácome Cunha, João Paulo Fernandes, and João Saraiva. 2020. SPELLing out energy leaks: Aiding developers locate energy inefficient code. *Journal Systems and Software* 161 (2020). https://doi.org/10.1016/j.jss.2019.110463

[33] Rui Pereira, Tiago Carção, Marco Couto, Jácome Cunha, João Paulo Fernandes, and João Saraiva. 2017. Helping Programmers Improve the Energy Efficiency of Source Code. In *Proceedings of the 39th International Conference on Software Engineering Companion* (Buenos Aires, Argentina) *(ICSE-C '17)*. IEEE Press, Piscataway, NJ, USA, 238–240. https://doi.org/10.1109/ICSE-C.2017.80

[34] Rui Pereira, Marco Couto, Francisco Ribeiro, Rui Rua, Jácome Cunha, Joã Paulo Fernandes, and Joã Saraiva. 2017. Energy Efficiency Across Programming Languages: How Do Energy, Time, and Memory Relate?. In *Proceedings of the 10th ACM SIGPLAN International Conference on Software Language Engineering* (Vancouver, BC, Canada) *(SLE 2017)*. ACM, New York, NY, USA, 256–267. https://doi.org/10.1145/3136014.3136031

[35] Rui Pereira, Marco Couto, Joã Saraiva, Jácome Cunha, and Joã Paulo Fernandes. 2016. The Influence of the Java Collection Framework on Overall Energy Consumption. In *Proceedings of the 5th International Workshop on Green and Sustainable Software (GREENS '16)*. ACM, 15–21.

[36] Rui Pereira, Pedro Simão, Jácome Cunha, and João Saraiva. 2018. jStanley: Placing a Green Thumb on Java Collections. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering* (Montpellier, France) *(ASE 2018)*. ACM, New York, NY, USA, 856–859. https://doi.org/10.1145/3238147.3240473

[37] Gustavo Pinto and Fernando Castor. 2017. Energy efficiency: a new concern for application software developers. *Commun. ACM* 60, 12 (2017), 68–75.

[38] Gustavo Pinto, Fernando Castor, and Yu David Liu. 2014. Mining questions about software energy consumption. In *Proceedings of the 11th Working Conference on Mining Software Repositories*. ACM, 22–31.

[39] Meikang Qiu, Zhong Ming, Jiayin Li, Keke Gai, and Ziliang Zong. 2015. Phase-Change Memory Optimization for Green Cloud with Genetic Algorithm. *IEEE Trans. Comput.* 64 (2015), 3528 – 3540.

[40] Rui Rua, Marco Couto, and João Saraiva. 2019. GreenSource: A Large-Scale Collection of Android Code, Tests and Energy Metrics. In *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*. 176–180.

[41] Roy Schwartz, Jesse Dodge, Noah Smith, and Oren Etzioni. 2019. Green AI. https://arxiv.org/abs/1907.10597.

[42] Damiano Torre, Giuseppe Procaccianti, Davide Fucci, Sonja Lutovac, and Giuseppe Scanniello. 2017. On the Presence of Green and Sustainable Software Engineering in Higher Education Curricula. In *Proceedings of the 1st International Workshop on Software Engineering Curricula for Millennials* (Buenos Aires, Argentina) *(SECM '17)*. IEEE Press, 54–60. https://doi.org/10.1109/SECM.2017.4

[43] Eelco Visser, Zine-el-Abidine Benaissa, and Andrew Tolmach. 1998. Building Program Optimizers with Rewriting Strategies. *SIGPLAN Not.* 34, 1 (Sept. 1998), 13–26.

[44] Ziliang Zong, Rong Ge, and Qijun Gu. 2017. Marcher: A Heterogeneous System Supporting Energy-Aware High Performance Computing and Big Data Analytics. *Journal of Big Data Research* 8 (2017), 27–38.