Energy Efficient Software in an Engineering Course

João Saraiva[1,2] and Rui Pereira[1]

[1] Department of Informatics, University of Minho
[2] HASLab/INESC TEC
Portugal
saraiva,rui.pereira@di.uminho.pt

Abstract. Sustainable development has become an increasingly important theme not only in the world politics, but also an increasingly central theme for the engineering professions around the world. Software engineers are no exception as shown in various recent research studies. Despite the intensive research on green software, today's undergraduate computing education often fails to address our environmental responsibility. We present a module on energy efficient software that we introduced as part of an advanced course on software analysis and testing. In this module we study techniques and tools to analyze and optimize energy consumption of software systems. Preliminary results of the first four instances of this course show that students are able to optimize the energy consumption of software systems.

Keywords: Sustainable Software Development, Energy Efficient Software

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 our Internet of Things (IoT) age 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, in this century energy consumption is becoming the main bottleneck when developing such systems [1].

Non wired/mobile devices are our everyday computers and not only do they need energy efficient hardware, but also need energy efficient software. While the

computer hardware manufacturers have for several decades already, done a considerable amount of research/work on developing energy efficient computers, only recently have the programming language and software engineering communities started conducting research on developing energy efficient software, the so-called green software. Indeed, green software is nowadays a very active research area, as shown by the organization of specific research events on this area (for example, the ICT4S and IGSC conferences, and the GREENS, RE4SuSy, and MeGSuS workshops), the many research papers being published in top conferences on, for example, green data structures [2–4],green software libraries [5], green rankings of software languages [6], energy greedy programming practices/patterns [7–10], and green repositories [11], etc.

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 [1, 12–14]. For example, in [14] a large survey on green software is presented with the following conclusions:

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

In fact, all those recent studies show that academia should not only define new advanced techniques and tools for green software developing, but it should also educate software engineers towards greener software development. Obviously, this education should be provided from the very beginning of a software engineer career. Unfortunately, today's undergraduate computing education often fails to address our social and environmental responsibility [15]. Indeed, energy efficiency and sustainability should be part of an undergraduate curriculum in software engineering.

This document presents a module on green software as part of a discipline on software engineering that is given in the MSc program on "Engenharia Informática" at University of Minho, Portugal. The course introduces green software as a new nonfunctional requirements in software engineering: minimizing software energy consumption is a key concern. Techniques to monitor, instrument and measure energy consumption by software systems are introduced. To understand the impact of how programmers develop software on energy consumption, we focus our energy analysis and optimization on the software's source code. Thus, a catalog of energy greedy programming practices is presented to students- that we call red smells - together with corresponding source code optimizations that reduce such possibly abnormal consumption, named green refactorings. Using this catalog, students have to build in a lab environment a successful experiment for software energy consumption. This paper also briefly presents our catalog of smells/refactorings, developed in the context of the Green Software Laboratory project [16]. Then, we present the first preliminary results obtained in three instances of this course, where students used this catalog to optimize the energy consumption of a given software system. The results show that students do understand the impact of software over energy consumption, are able

to locate red smells in its source code and to apply appropriate refactorings to optimize the software as to reduce energy consumption.

2 Energy Efficient Software in Higher Education

2.1 Sustainable Development and its Dimentions

Over the past decade, interest in topics related to sustainability has grown steadily. In fact, sustainability is studied from various angles, for example, economic, political, institutional, cultural or ethical ones, and thus it is difficult to arrive at a common definition that covers nearly all of its aspects. The broader socio-political concept of sustainability, as presented by the World Commission on Environment and Development in their 1987 report [17], addresses the well-known conflicts between environment and development goals by formulating sustainability *"as the ability to make development sustainable to ensure that it meets the needs of the present without compromising the ability of future generations to meet their own needs."*.

In the extensive discussion and use of this concept, it is commonly agreed that the central component of sustainable development is best described by considering the following three dimensions of sustainability [18]:

- *Economic:*: An economically sustainable system must be able to produce goods and services, and to avoid extreme sectoral imbalances which damage agricultural or industrial production.
- *Environmental:* An environmentally sustainable system must maintain a stable resource base, avoiding over-exploitation of renewable resources, and depleting non-renewable resources only to the extent that investment is made in adequate substitutes.
- *Social:* A socially sustainable system must achieve distributional equity, adequate provision of social services health and education, gender equity, and political accountability and participation.

2.2 Sustainable Development in Higher Education

There are several works studying how to integrate the concept of sustainable development into higher education in meaningful ways and to address the three main dimensions of sustainability and their different combinations [19–21]. For example, at the university institutional level, [20] presents a procedure at the University of Gävle in Sweden, designed to stimulate integration of the concept of sustainable development into courses and research projects. In that study, faculty members were asked to classify their courses and research funding applications regarding their contributions to sustainable development. The method of classifying courses provides a framework to approach sustainable development from common definitions, but still allows for individual approaches to integrating it in courses and research. The study shows that it is possible to integrate the concept of sustainable development into higher education. However, the system

needs further development in order to show instructors and researchers that the integration of sustainability is seen as important to the university administration so that it stimulates faculty members to integrate sustainable development in their courses.

At the national level, the paper [19] describes the process of promoting the disciplinary exploration of sustainable development in the curricula of Dutch Universities and the lessons learned with that process. The Finnish Government included the promotion of sustainable development in its development plan for education and research in 2003. This development plan is a key steering tool for the Finish ministry of education [22]. In 2010, eight public universities in Hong Kong signed the Hong Kong Declaration to recognize the vital role of the higher education sector in the efforts to deal with the challenges caused by climate change and to include the collective voices from Hong Kong public universities in the global sustainability community. In [23] a study reviewed what the eight public universities in Hong Kong have accomplished in promoting sustainability.

Sustainable development is also a concern for worldwide institutions, like for example ACM: the world's largest association of computing professionals. As advocated in [24] information systems can be a driving force for sustainability improvements and, as a consequence, ACM members could and should play a critical role in creating and implementing an information strategy. In fact, ACM promotes sustainability and in 2017, SIGPLAN[3] formed the climate committee to study the climate impact of conferences and possible steps that SIGPLAN might take in response. ACM hosts a number of annual scientific meetings at various locations throughout the world. While such meetings are important for furthering important research, the air travel required for participate in such meetings is a significant source of greenhouse gas emissions, which in turn are a significant contributor to environmental change. In their preliminary report on *Engaging with Climate Change: Possible Steps for SIGPLAN* , the climate committee (among other things) presents a number of alternative models such a physical/virtual hybrids, multi-hub conferences, and regional conferences, with the goal to reducing carbon footprints. To make its members aware of the conference participation impact on climate, ACM also provides a CO_2 footprint calculator for conferences: `https://co2calculator.acm.org/`.

Sustainability in the ACM Computer Science Curricula While technical issues are central to the computing curriculum, they do not constitute a complete educational program in the field. Students must also be exposed to the larger societal context of computing to develop an understanding of the relevant social, ethical, legal and professional issues.

Sustainability was first introduced in the ACM Computer Science 2008 curricular guidelines, and in that same year [25] presented a policy on computing education for sustainability for adoption by ACM SIGCSE[4]. In the Computer

[3] The ACM Special Interest Group on Programming Languages
[4] The ACM Special Interest Group on Computer Science Education

Science Curricula 2013, ACM further recognize the enormous impact that computing has had on society at large emphasizing a sustainable future and placing added responsibilities on computing professionals.

The outcome of this was the definition of core topics on sustainability which include the identification of ways to be a sustainable practitioner by taking into consideration cultural and environmental impacts of implementation decisions (e.g. organizational policies, economic viability, and resource consumption). The exploration of global social and environmental impacts of computer use and disposal (e-waste). And, the assessment the environmental impacts of design choices in specific areas such as algorithms, operating systems, networks, databases, or human-computer interaction.

2.3 Energy Efficient Software in Higher Education

In recent years, sustainable and energy efficient (green) software became a very active software engineering research field. However, there are several studies showing that software engineers still lack knowledge of how to reason and improve energy consumption of their software systems.

In [13] a detailed study on energy-related questions on Stack-Overflow - a question and answer site for (non) professional software developers[5] - showed that software developers are aware of the energy consumption problems but the many questions they asked rarely got appropriate answers. They also suggested eight strategies to reduce energy consumption through software modification. A large empirical study of how developers think about energy when they write requirements, design, construct, test, and maintain their software is presented in [12]. After surveying 464 developers (from ABB, Google, IBM, and Microsoft) and 18 in-depth interviews with Microsoft employees the study overall conclusions are: "green software engineering practitioners care and think about energy when they build applications; however, they are not as successful as they could be because they lack the necessary information and support infrastructure.". In [14] show that programmers know little about energy consumption: "The programmers in our study lacked knowledge and awareness of software energy-related issues. More than 80 percent of them didn't take energy consumption into account when developing software.". And, authors suggest that the strategies discussed in [13] "should be part of programmers' education. In addition, development tools can be created to identify unnecessary energy consumption and suggest how to reduce it. Educators could develop slides, videos, projects, and assignments as part of an undergraduate curriculum for energy efficiency and sustainability.". A recent article at CACM [1] discusses energy efficiency as a new concern for software developers, showing that: developers currently do not know how to write, maintain and evolve green software. They lack the knowledge on how to measure, profile and optimize energy consumption, and they lack tools to help them in these tasks.

[5] `http://stackoverflow.com`

Despite the intensive research on green software and all these recent studies showing the lack of knowledge and language/tool support software engineers are currently facing, there is little undergraduate computing education in green software engineering. Misconception among developers and researchers persist, rooted in a lack of coherent understanding of sustainability, and how it relates to software systems research and practice, also makes it difficult [26]. [26] presents a cross-disciplinary initiative to create a common ground and a point of reference for the global community of research and practice in software and sustainability, to be used for effectively communicating key issues, goals, values and principles of sustainability design for software-intensive systems.

Nevertheless, there is some work advocating the introduction of sustainability in undergraduate education. In [27] a first study of what is the current state of teaching sustainability in the software engineering community is presented. The paper reports the findings from a targeted survey of 33 academics on the presence of green and sustainable software engineering in higher education. The major findings suggest that sustainability is under-represented in the curricula and the main reasons are:

- lack of awareness,
- lack of teaching material,
- high effort required,
- lack of technology and tool support.

A list a group of barriers for sustainability integration into computing education is also discussed in [15]. Two strategies to sustainability integration in computing are presented: the developing of a new course or the development of modules easily plugged into existing courses. This short paper gives a very general view of the organization of such a course. However, it does not include any discussion on the course objectives and whether they were achieved by students or not.

A systematic approach for teaching software engineering for sustainability and its qualitative evaluation is presented in [28]. The proposed course blueprint articulated a candidate set of modules. This is an intensive week-long course, given at a summer school where participants had different backgrounds. In [29] a course on learning and teaching computing sustainability is also briefly described. They are adapting an existing course on professionalism in computing to incorporate more of these sustainability modules, such as: green mobile cloud computing systems; integration of green clouds and the Internet of things; energy saving solutions and trade-offs; sensors and monitoring software tools for evaluating energy use, among other topics. [30] presents an experiment with integrating issues of sustainability with information technology in both introductory and upper-level computer science courses. The course discusses several case studies that illustrate the many creative ways that IT is being used to address sustainability: transportation and logistics, supply chain management, etc. In [21] it is described the design of the course *Software Engineering Sustainability* that introduces the sustainable concept into educational programs for software engi-

neers. The course has been being delivered at the National Aerospace University "Kharkiv Aviation Institute" in Ukraine.

At the Texas State University in USA, Ziliang Zong offers a full course on Advanced Green Computing[6] which covers hardware and software techniques to improve the energy-efficiency of computing systems. Topics include best practices in building energy-efficient data centers and mobile devices, current trends in reducing the energy consumption of processors and storage components, energy-aware resource management, software optimizations, and hands-on experience on power-measurable systems. The objectives of this course is that students will be able to:

- Analyze research papers and evaluate existing research ideas.
- Compare experimental results from different algorithms.
- Evaluate the strengths and weaknesses of different approaches.
- Design experiments and collect experimental results on power measurable systems

Patricia Lago and Ivano Malavolta coordinate the Master track in Software Engineering and Green IT at the Master's in Computer Science[7] offered at Vrije Universiteit Amsterdam, The Netherlands. The combination of Software Engineering and Green IT in one track provides the students with the instruments necessary to gain a holistic understanding of large-scale and complex software systems, to manage their evolution, assess their quality and environmental impact, quantify their value and sustainability potential, and organize their development in different local and distributed contexts. Students graduating in this track are experts of:

- Architecture design of software-intensive systems
- the role of software for sustainability (including energy efficiency, socio-technical, ecologic and economic impact)
- software engineering techniques for critical analysis and decision-making
- benefits and challenges of developing and maintaining sustainable software
- the pervasive role of software-intensive systems in the digital society
- data-driven measurement and assessment of software quality

3 Software Analysis and Testing with a Green Flavor

In this section we discuss in detail the module on green software we offer as part of a non mandatory discipline on software engineering, more precisely on software analysis and testing, that is given in the fourth year of our five year MSc program.

As mentioned in several papers advocating the inclusion of sustainability and green software in computing education [26–28, 31–34], we present green software

[6] CS 7333 - Advanced Green Computing, `https://cs.txstate.edu/academics/course_detail/CS/7333/`.

[7] `https://vuweb.vu.nl/en/education/master/computer-science`.

as a module of an already existing course. Moreover, we focus this module in making future software engineers aware of the impact of programming practices on software energy consumption.

3.1 Green Software: a Multidisciplinary Module

The green software module requires a multidisciplinary course combining several software engineering techniques and principles, namely:

Source Code Analysis and Transformation: In order to analyze and transform software systems we introduce two powerful source code manipulation techniques: Strategic and Aspect Oriented Programming. Strategic programming is a generic tree traversal techniques that allows for expressing powerful abstract syntax tree analysis and transformations. Aspect oriented programming is introduced to allow developers to instrument the base source code without adding the energy monitoring intrusive code, but keeping it in one aspect that is later weaved to the base program.

Green Aspect: In order to monitor the energy consumption, students need to traverse and instrument the source code with calls to APIs providing energy measurements at runtime. In our course we consider two types of measurements: energy estimation provided by manufacturers of the CPUs, namely the RAPL framework developed by Intel [35], or using hardware with energy sensors, like for example the ODroid hardware board[8].

Source Code Smells and Metrics: Code smells represent symptoms of poor implementation choices when developing software. Code smells are not faults, they make program understanding difficult, and possibly indicate a deeper problem in the software. Software metrics are usually used to detect source code smells, for example, a too long method smell.

Green Aspect: In our module on green software we present a catalog of energy greedy programming practices [10]. This catalog can also be seen as a energy smell catalog, where software metrics can be used to detect such smells in the source code.

Program Refactoring: refactoring is a controlled source-to-source transformation technique for improving the design of an existing (source code) software system. Its essence is applying a series of small semantic-preserving transformations. Refactorings are usually associated with code smells: for each smell there is a refactoring that eliminates it.

Green Aspect: We associate refactorings to the catalog of energy smells so that students can use a green refactoring to eliminate red smells. Because the main focus of refactoring is to improve comprehensibility, several refactorings may negatively affect energy consumption. Students also analyze how refactorings available from Java IDEs affect energy consumption. The catalog of green refactorings for Java data structures in supported by the jStanley tool tool [2].

[8] `http://www.odroid.com`

Technical Debt: Technical debt describes the gap between the current state and the ideal state of a software system. The key idea of technical debt is that software systems may include hard to understand/maintain/evolve artefacts, causing higher costs in the future development and maintenance activities. These extra costs can be seen as a type of debt that developers owe the software system.

Green Aspect: In our module we introduce the concept of Energy Debt [36] as the amount of unnecessary energy that a software system uses over time, due to maintaining energy code smells for sustained periods.

Software Testing and Benchmarking Infrastructures: Software testing aims at ensuring that a software system is defect free. We present the usual levels of testing: unit, integration, system, regression and beta testing. Automated test case generation and property based testing is also studied in this course. Code coverage and mutation-based testing is used to assess the quality of the test suite. Moreover, we use testing framework and benchmarks infrastructures, like Google's Caliper[9] in order to execute programs.

Green Aspect: To measure energy consumption, the source code needs to be executed with proper inputs. We use system testing, where the automated test case generation techniques produces *real* inputs of the program under testing.

Fault Localization: When a software systems fails running the defined/generated test suite, programmers need to locate the fault and fix it. Spectrum-based Fault Localization (SFL) relies on test cases to run the program and it uses statistical methods to assign probabilities of being faulty to source code components (methods, classes, statements, etc).

Green Aspect: Abnormal energy consumption can be seen as a software fault. In our course we defined a variant of SFL to locate energy leaks in the source code: Spectrum-based Energy Leak Localization (SPELL) [37, 38], and students can use it to locate such energy hot-spots in their software.

Automated Program Repair: The goal of automated program repair is to take a faulty program and a test suite, and automatically produce a patch that fixes the program. The test suite provides the correctness criterion in this case, guiding the repair towards a valid patch.

Green Aspect: SPELL adapts fault localization to the green software realm, while green refactorings eliminate red smells aiming at improving energy efficiency of programs. We combine these two techniques in order to automate the energy-aware repair of energy inefficient software systems [39, 40].

3.2 Green Software: Module Objectives

The objectives of the green software module are:

[9] `http://code.google.com/p/caliper/`

- 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.

Course Duration, Organization and Evaluation: The module of green software is part of the software analysis and testing course. This course one semester long course with 5 ECTS. It is a non mandatory course included in the the second semester fourth of the year of the master program on software engineering at Minho University.

The students have 3 hours per week in the classroom: one hour in a seminar room, where all theories and techniques are presented. The remaining two weekly hours are laboratory classes where students have the chance to experiment the introduced techniques for software energy consumption. The evaluation consists of two components: an individual written exam, and a group project on analyzing and optimizing the energy consumption of a given software system. The considered software system is the students project developed in the introductory course to object oriented programming the semester before (by second year students). The idea is to provide students in the course with a simple, non fully optimized system.

In order to analyze and optimize the energy consumption of Java based software systems, we present the students a catalog of energy-greedy Java programming practices. The main goal is to make students aware of some features of Java's source code that may indicate an abnormal energy consumption of the software. The students are also presented with a possible solution by performing a refactoring of the source code into a more energy efficient one. Moreover, software tools that locate such features and (semi) automatically optimize the code are also presented. In laboratory sessions the students are able to experiment with smell detection and optimization. Then, outside of class, students have to work in group (three students per group) and apply the catalog/tools in order to optimize the energy consumption of a real software project.

3.3 Green Software: Module Supporting Tools

In the context of the Green Software Laboratory project [16, 41] we have developed several tools that support the laboratory classes, namely:

- *SPELL* [38]- A toolkit to measure the energy consumption of a Java based program and detect potential energy hot spots through an adapted Spectrum-based Fault Localization technique.
- *jStanley* [2] - An Eclipse plugin that automatically refactors Java collections to more energy efficient ones.
- *Chimera* [10]: An energy-greedy Android pattern testing framework for Android.

– *E-Debitum* [42] - A SonarQube extension to manage the Energy Debt of Java/Android-based software systems.
– *GreenSource* and *AnaDroid* [11] - A repository of android source code applications tailored for green software analysis and a tool to static analyze and dynamically monitor the energy consumption of such applications.

4 Energy Efficent Software: Students Assessment

We introduced this module on green software as part of the software analysis and testing course in the scholar year of 2017/2018. In the 2020/2021 scholar year we offered the fourth instance of the course, totaling 141 students enrolled across all four instances.

In the first four instances of this module students were quite positive in their reception of the material and the way it was incorporated in the course. In fact across all editions, 59% of students obtained at least a B for the module project, with the average module grade (written exam and project) being a C.

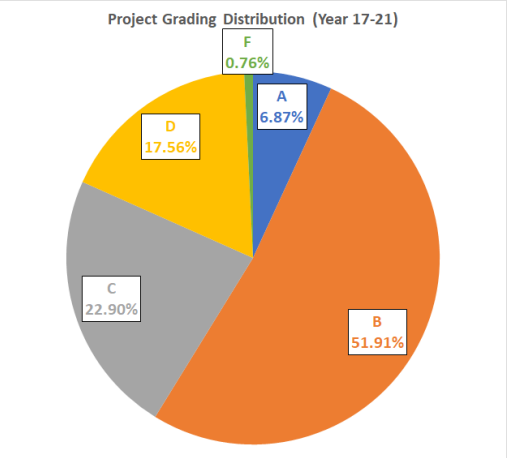

Fig. 1: Project grade distribution across all editions

As shown in Figure 1, most students received a positive evaluation on the group project where they have to analyze and optimize the energy consumption of a given software system. With the green software background acquired in the course, students were not only able to measure energy consumption of a software system, but also to optimize its energy consumption. Indeed, the catalog of energy smells and corresponding green refactorings, introduced in the theoretical classes, provided insights how energy may be abnormally consumed by software and pointed to the exact locations of where to improve/refactor the source code.

In order to assess the green software learning outcomes of individual students, the written exam includes questions on green software, as well. In the 2018-2019

scholar year we defined a specific question in regards green software. This one question has been repeated in the future instances so that we have a larger set of answers.

In this exam question on green software, students are asked to identify energy smells in the source code of a given Java class (approx. 100 lines long), and to refactor each of the identified smells by hand. The given Java source code contains six of the smells included in the red smells catalog we present in the theoretical classes. The following subsection presents a brief description of the red smells catalog, and in Section 4.2 we analyze the students' answers in detail .

4.1 A Catalog of Energy Smells and Refactorings

In the theoretical classes we present a catalog of Java-based energy smells and associated refactorings reported in the green software literature [4, 43, 44]. The following lists the subset of smells that occur in the given Java source code for the exam question.

Data Structures: Most languages offer mechanisms to manipulate data structures. Java is no exception with the Java Collections Framework (JCF). There are several research papers analyzing the energy behavior of such Java structures [2–4, 45] showing very different energy efficiencies.

String Manipulation: Strings are widely used when developing software, with modern languages providing special syntax/operators to manipulate them. Java uses the String class and includes the "+" operator to concatenate them. However, the `StringBuilder` class in the Java library exploits buffering, and is more energy efficient. Thus, every occurrence of the string concatenation "+" in the source code is an energy smell and it should be refactored to a `StringBuilder`.

Lambda Expressions: Java 8 adopted lambda expressions as a mechanism to manipulate its collections. However, the execution of Java streams has several efficiency problems, either by doing more traversals than necessary, or creating intermediate data structures. In fact, Java 8 streams are still an order of magnitude slower than hand-written loops [43, 46, 47]. Thus, the use of Java streams is an energy smell and the refactoring considers a for-loop instead. IntelliJ IDEA[10] refactoring source code system provides such a detection and refactoring.

Accessing Object Fields: The object oriented paradigm encourages encapsulation, in order to make sure that "sensitive" data is hidden from users. Thus, every class should provide public getters and setters. However, the overhead caused by often calling getters and setters can increase both the execution time and energy consumption. Thus, the use of a get/set method is an energy smell and can be refactored to direct access of the attribute [44].

[10] `https://www.jetbrains.com/idea/`

Java Exceptions: Exceptions are used to manage any unexpected event in the code, while ensuring code readability. When an object is in a condition it cannot handle, it raises an exception to be captured by another object. The JVM searches backward through the call stack to find methods that do can handle the exception. Exception handling is expensive and involves object creation, thus it should be avoided by, for example, the methods returning error codes [44].

4.2 Students Grades

Figure 2 shows the individual grades that students received on the specific question regarding energy smell detection and refactoring the given Java class aiming at improving its energy consumption.

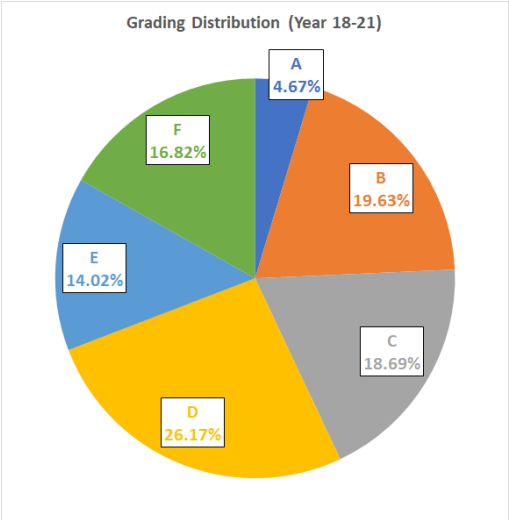

Fig. 2: Green Software Question: Distribution of grades.

As shown in Figure 2, a quarter of the students received a very good (19.63%) or excellent (4.67%) grade. On the other end, 16.82% of the students failed in answering this question. Most of these students (60%) also performed poorly in the other questions and did fail in the overall exam (as we can see in Figures 4, 5, and 6).

The source code of the Java class included in the question contains five energy smells from the (larger) catalog we introduce in the green software module. Figure 3 shows the percentage of students who identified each of the occurring energy smells, in each instance of the course.

As we can observe, when we consider the three instances of the exam, the *String Manipulation* smell was identified by roughly 80% of the students, while the *Accessing Object Fields* smell (shown as Gets/Sets) was detected by less than

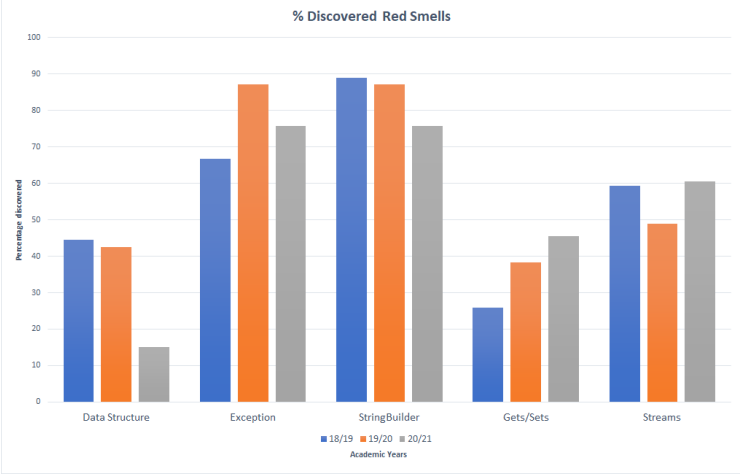

Fig. 3: Percentage of students discovering each red smell

30%, only. Such a large difference in smell detection has two possible answers: since the very first Java OO programming course, students are taught to avoid the inefficient pre-defined Java string concatenation, thus they are familiar with identifying it and eliminating it. On the other end, the OO paradigm teaches/-motivates encapsulation, and thus the use of gets and sets to access the (private) state of an object. As a consequence, students do not find such a refactoring natural. Similarly, replacing the elegant Java exceptions mechanism by the use of an (C-like) error code also goes against the OO paradigm. These students, however, do have a strong background in imperative programming (with the C language) and, thus, are used to this basic style of handling exceptions.

Streams offer an advanced functional style of programming in Java. Although students have a good background in functional programming (in Haskell), most students find it hard to understand the concept of higher-order functions and to adopt it. As a result, only half of the students were able to understand, detect and refactor this Java energy smell. Although the transformation from a stream to a for-loop is simple[11], students are not able to fully understand the functional code and to hand-write such refactoring themselves.

The students also performed poorly in detecting and refactoring Java energy greedy collections: in the three instances less than 40% correctly answer this question. Moreover, in the last instance the grades dropped to approximately 15%. While many of the other energy greed smells taught within the course are relatively straightforward rules (i.e. replacing string concatenation with StringBuilders), choosing the most energy efficient data structure is inherently more difficult. This is due to having a wide array of choices between the different collections, which also depends on the needed methods and operations. Adding

[11] Actually, modern Java IDEs, such as IntelliJ, offer this transformation as a predefined refactoring

another requirement, in this case energy efficiency, further raises the complexity. While during the practical lab sessions students had tools and/or lecture slides at their disposal, many chose to not take the lecture material to the exam.

Figures 4, 5 and 6 show the comparison between the overall grade students obtained in the individual exam (represented as bars with the left-axis) and on the green software question (represented as lines with the right-axis), in the three instances completed.

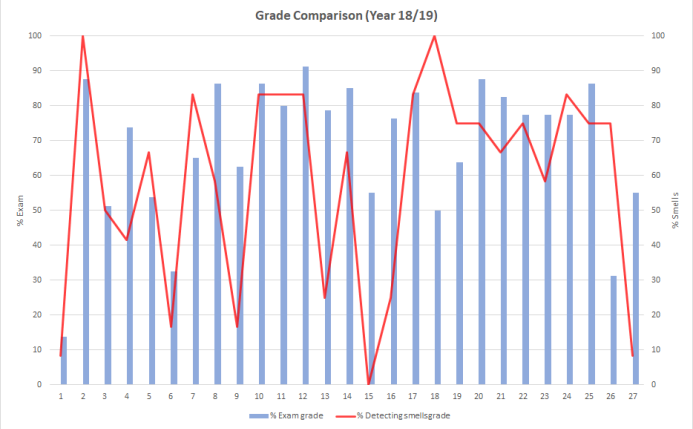

Fig. 4: Comparison between exam grade and red smell detection grade for academic year 2018/2019

We can see that only 8 out of 27 obtained a failing score in the 2018/2019 instances. The average score of the full exam is 68.6%, while the average of the green software question is 58.6%. We can also observe in Fig. 4 that students who performed well in the full exam, also received a good make in the green software question. Additionally, during this first instance, two students obtained a perfect score for this question (100% grade).

In the 2019/2020 instance of the course, 11 out of 46 failed answering the exact same green software question. When we compare the scores to the previous instance, we observe that both the average of the exam and the question decreased: 57.6% and 55.7%, respectively. However, we observe the same pattern in the results: students who performed well in the full exam, also performed well in the question. This is also the case for the 2020/2021 edition.

In the latest instance of the course, 7 out of 33 failed in the green software question. The average scores went up and are similar to the results of the first instance: the exam score is 64.7%, that is also similar to the average of the green software questions 60.1%.

The second instance registered the most number of students in this non-mandatory course, having 70% more students than the first edition, and 40% more than the most recent edition. It is in this second instance that students

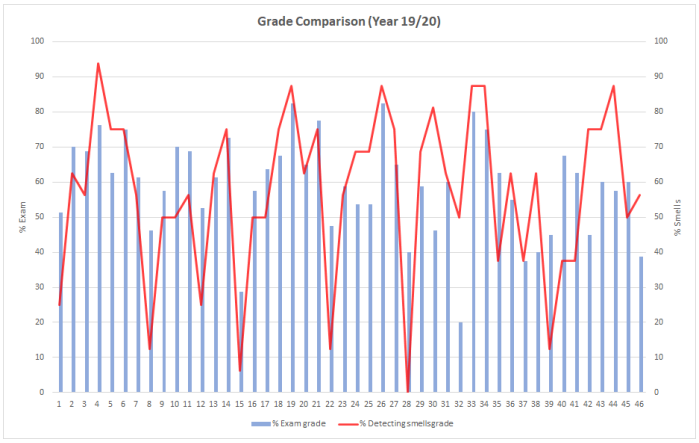

Fig. 5: Comparison between exam grade and red smell detection grade for academic year 2019/2020

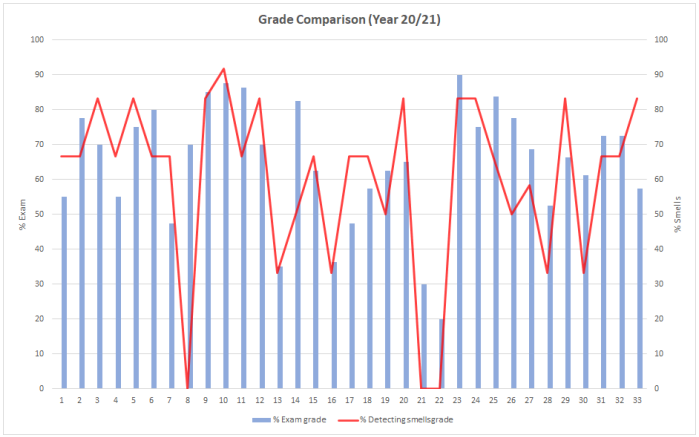

Fig. 6: Comparison between exam grade and red smell detection grade for academic year 2020/2021

received the lowest average scores, both in terms of the full exam and the green software question. Since this is a non-mandatory course, the increase in numbers not only influences (negatively) the quality of students, but also makes it harder to teach an advanced course to a larger group of students. We are convinced that this is the main reason for the worsened performance of the students in that instance. In fact, if we consider the results of the 30 best students of 2018/2019, a number similar to the previous/following instance, then the average scores are 66.9% (exam) and 65.4% (question), which in terms of the green software question would the be the best average result to date.

5 Conclusions

This paper presented the module on green software that we introduced as part of the course on software analysis and testing: an advanced course on software engineering. We described in detail the green flavor we incorporated in this well established multi-disciplinary course. Furthermore, we have assessed students in green decision making when developing/optimizing energy efficient software. Our first preliminary results are positive: students acquired the necessary engineering skills to measure and optimize energy consumption of software systems.

Aknowlegments

This work is financed by the ERDF European Regional Development Fund through the Operational Programme for Competitiveness and Internationalisation - COMPETE 2020 Programme within project POCI-01-0145-FEDER-006961, and by National Funds through the Portuguese funding agency, FCT - Fundação para a Ciência e a Tecnologia within project POCI-01-0145-FEDER-016718 and UID/EEA/50014/2013.

References

1. G. Pinto, F. Castor, Energy efficiency: a new concern for application software developers, Communications of the ACM 60 (12) (2017) 68–75.
2. R. Pereira, P. Simão, J. Cunha, J. Saraiva, jstanley: Placing a green thumb on java collections, in: Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, ASE 2018, ACM, New York, NY, USA, 2018, pp. 856–859. doi:10.1145/3238147.3240473.
URL http://doi.acm.org/10.1145/3238147.3240473
3. S. Hasan, Z. King, M. Hafiz, M. Sayagh, B. Adams, A. Hindle, Energy profiles of java collections classes, in: Proceedings of the 38th International Conference on Software Engineering, ACM, 2016, pp. 225–236.
4. R. Pereira, M. Couto, J. Saraiva, J. Cunha, J. P. Fernandes, 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, 2016, pp. 15–21.
5. M. Linares-Vásquez, G. Bavota, C. Bernal-Cárdenas, R. Oliveto, M. Di Penta, D. Poshyvanyk, Mining energy-greedy api usage patterns in android apps: an empirical study, in: Proceedings of the 11th Working Conference on Mining Software Repositories, ACM, 2014, pp. 2–11.
6. R. Pereira, M. Couto, F. Ribeiro, R. Rua, J. Cunha, J. P. Fernandes, J. Saraiva, 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, SLE 2017, ACM, New York, NY, USA, 2017, pp. 256–267. doi:10.1145/3136014.3136031.
URL http://doi.acm.org/10.1145/3136014.3136031

7. L. Cruz, R. Abreu, Performance-based guidelines for energy efficient mobile applications, in: Proceedings of the 4th International Conference on Mobile Software Engineering and Systems, MOBILESoft '17, IEEE Press, Piscataway, NJ, USA, 2017, pp. 46–57. doi:10.1109/MOBILESoft.2017.19.
URL https://doi.org/10.1109/MOBILESoft.2017.19
8. D. Li, W. G. J. Halfond, An investigation into energy-saving programming practices for android smartphone app development, in: Proceedings of the 3rd International Workshop on Green and Sustainable Software, GREENS 2014, ACM, New York, NY, USA, 2014, pp. 46–53. doi:10.1145/2593743.2593750.
URL http://doi.acm.org/10.1145/2593743.2593750
9. R. Morales, R. Saborido, F. Khomh, F. Chicano, G. Antoniol, Earmo: An energy-aware refactoring approach for mobile apps, IEEE Transactions on Software Engineering 44 (12) (2018) 1176–1206.
10. M. Couto, J. Saraiva, J. P. Fernandes, Energy refactorings for android in the large and in the wild, in: 2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER), 2020, pp. 217–228.
11. R. Rua, M. Couto, J. Saraiva, GreenSource: A large-scale collection of android code, tests and energy metrics, in: 2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR), 2019, pp. 176–180.
12. I. Manotas, C. Bird, R. Zhang, D. Shepherd, C. Jaspan, C. Sadowski, L. Pollock, J. Clause, An empirical study of practitioners' perspectives on green software engineering, in: Proceedings of the 38th International Conference on Software Engineering, ICSE '16, Association for Computing Machinery, New York, NY, USA, 2016, p. 237–248. doi:10.1145/2884781.2884810.
URL https://doi.org/10.1145/2884781.2884810
13. G. Pinto, F. Castor, Y. D. Liu, Mining questions about software energy consumption, in: Proceedings of the 11th Working Conference on Mining Software Repositories, ACM, 2014, pp. 22–31.
14. C. Pang, A. Hindle, B. Adams, A. E. Hassan, What do programmers know about software energy consumption?, IEEE Software 33 (3) (2016) 83–89.
15. Y. Cai, Integrating sustainability into undergraduate computing education, in: Proceedings of the 41st ACM Technical Symposium on Computer Science Education, SIGCSE '10, Association for Computing Machinery, New York, NY, USA, 2010, p. 524–528. doi:10.1145/1734263.1734439.
URL https://doi.org/10.1145/1734263.1734439
16. Green Software Laboratory.
URL http://greenlab.di.uminho.pt/
17. G. H. Brundtland, Our common future, from one earth to one world - an overview by the world commission on environment and development (march 1987).
URL https://sustainabledevelopment.un.org/content/documents/5987our-common-future.pdf
18. J. Harris, Basic Principles of Sustainable Development, in: S. K. Bawa, R. Seidler (Eds.), Dimensions of Sustainable Development, Vol. 1, Encyclopedia of Life Support Systems - EOLSS, Oxford, United Kingdom, 2009, Ch. 2, pp. 21–40.
19. G. Appel, I. Dankelman, K. Kuipers, Disciplinary Explorations of Sustainable Development in Higher Education, Springer Netherlands, Dordrecht, 2004, pp. 213–222. doi:10.1007/0-306-48515-X_16.
URL https://doi.org/10.1007/0-306-48515-X_16
20. K. Sammalisto, T. Lindhqvist, Integration of sustainability in higher education: A study with international perspectives, Innovative Higher Education 32 (2008) 221–233.

21. I. Turkin, Y. Vykhodets, Software engineering master's program and green it: The design of the software engineering sustainability course, in: 2018 IEEE 9th International Conference on Dependable Systems, Services and Technologies (DESSERT), 2018, pp. 662–666.
22. T. Kaivola, L. Rohweder (Eds.), Towards sustainable development in higher education - reflections, no. 2007:6 in Opetusministeriön julkaisuja, Opetusministeriö, koulutus- ja tiedepolitiikan osasto, Finland, 2007.
23. W. Xiong, K. H. Mok, Sustainability practices of higher education institutions in hong kong: A case study of a sustainable campus consortium, Sustainability (2) (2020) 452. doi:https://doi.org/10.3390/su12020452.
24. R. T. Watson, J. Corbett, M. C. Boudreau, J. Webster, An information strategy for environmental sustainability, Commun. ACM 55 (7) (2012) 28–30. doi:10.1145/2209249.2209261.
URL https://doi.org/10.1145/2209249.2209261
25. S. Mann, L. Smith, L. Muller, Computing education for sustainability, SIGCSE Bull. 40 (4) (2008) 183–193. doi:10.1145/1473195.1473241.
URL https://doi.org/10.1145/1473195.1473241
26. C. Becker, R. Chitchyan, L. Duboc, S. Easterbrook, B. Penzenstadler, N. Seyff, C. C. Venters, Sustainability design and software: The karlskrona manifesto, in: Proceedings of the 37th International Conference on Software Engineering - Volume 2, ICSE '15, IEEE Press, 2015, p. 467–476.
27. D. Torre, G. Procaccianti, D. Fucci, S. Lutovac, G. Scanniello, 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, SECM '17, IEEE Press, 2017, p. 54–60. doi:10.1109/SECM.2017.4.
URL https://doi.org/10.1109/SECM.2017.4
28. B. Penzenstadler, S. Betz, C. C. Venters, R. Chitchyan, J. Porras, N. Seyff, L. Duboc, C. Becker, Everything is interrelated: Teaching software engineering for sustainability, in: Proceedings of the 40th International Conference on Software Engineering: Software Engineering Education and Training, ICSE-SEET '18, Association for Computing Machinery, New York, NY, USA, 2018, p. 153–162. doi:10.1145/3183377.3183382.
URL https://doi.org/10.1145/3183377.3183382
29. M. Hamilton, Learning and teaching computing sustainability, in: Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education, ITiCSE '15, Association for Computing Machinery, New York, NY, USA, 2015, p. 338. doi:10.1145/2729094.2754850.
URL https://doi.org/10.1145/2729094.2754850
30. K. Abernethy, K. Treu, Integrating sustainability across the computer science curriculum, J. Comput. Sci. Coll. 30 (2) (2014) 220–228.
31. C. Pattinson, Ict and green sustainability research and teaching, IFAC-PapersOnLine 50 (1) (2017) 12938 – 12943, 20th IFAC World Congress. doi:https://doi.org/10.1016/j.ifacol.2017.08.1794.
URL http://www.sciencedirect.com/science/article/pii/S2405896317324151
32. K. R. Berntsen, M. R. Olsen, N. Limbu, A. T. Tran, R. Colomo-Palacios, Sustainability in software engineering - a systematic mapping, in: J. Mejia, M. Muñoz, Á. Rocha, T. San Feliu, A. Peña (Eds.), Trends and Applications in Software Engineering, Springer International Publishing, Cham, 2017, pp. 23–32.
33. N. Wolfram, P. Lago, F. Osborne, Sustainability in software engineering, in: 2017 Sustainable Internet and ICT for Sustainability (SustainIT), 2017, pp. 1–7.

34. C. Calero, M. Piattini, Green in Software Engineering, Springer Publishing Company, Incorporated, 2015.
35. H. David, E. Gorbatov, U. R. Hanebutte, R. Khanna, C. Le, Rapl: memory power estimation and capping, in: International Symposium on Low-Power Electronics and Design (ISLPED), 2010 ACM/IEEE, IEEE, 2010, pp. 189–194.
36. M. Couto, D. Maia, J. Saraiva, R. Pereira, On energy debt: Managing consumption on evolving software, in: Proceedings of the 3rd International Conference on Technical Debt, TechDebt '20, Association for Computing Machinery, New York, NY, USA, 2020, p. 62–66. doi:10.1145/3387906.3388628.
URL https://doi.org/10.1145/3387906.3388628
37. R. Pereira, T. Carção, M. Couto, J. Cunha, J. P. Fernandes, J. Saraiva, Helping programmers improve the energy efficiency of source code, in: Proceedings of the 39th International Conference on Software Engineering Companion, ICSE-C '17, IEEE Press, Piscataway, NJ, USA, 2017, pp. 238–240. doi:10.1109/ICSE-C.2017.80.
38. R. Pereira, T. Carção, M. Couto, J. Cunha, J. P. Fernandes, J. Saraiva, Spelling out energy leaks: Aiding developers locate energy inefficient code, J. Syst. Softw. 161 (2020). doi:10.1016/j.jss.2019.110463.
URL https://doi.org/10.1016/j.jss.2019.110463
39. R. Pereira, M. Couto, F. Ribeiro, R. Rua, J. Saraiva, Energyware analysis, in: 7th Workshop on Software Quality Analysis, Monitoring, Improvement, and Applications (SQAMIA), Vol. 2217, CEUR Workshop Proceedings, 2018.
40. R. Pereira, Energyware engineering: Techniques and tools for green software development, Ph.D. thesis, Universidade do Minho (2018).
41. J. Saraiva, R. Abreu, J. Cunha, J. P. Fernandes, GreenSoftwareLab: Towards an engineering discipline for green software, Impact 2018 (1) (March 2018). doi:https://doi.org/10.21820/23987073.2018.9.
42. D. Maia, M. Couto, R. Pereira, J. Saraiva, E-Debitum: Managing software energy debt, in: 1st International Workshop on on Sustainable Software Engineering (SUSTAIN-SE), (to appear), 2020.
43. O. Kiselyov, A. Biboudis, N. Palladinos, Y. Smaragdakis, Stream fusion, to completeness, in: Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2017, Association for Computing Machinery, New York, NY, USA, 2017, p. 285–299. doi:10.1145/3009837.3009880.
URL https://doi.org/10.1145/3009837.3009880
44. M. Longo, A. Rodriguez, C. Mateos, A. Zunino, Reducing energy usage in resource-intensive java-based scientific applications via micro-benchmark based code refactorings, Comput. Sci. Inf. Syst. 16 (2) (2019) 541–564. doi:10.2298/CSIS180608009L.
URL https://doi.org/10.2298/CSIS180608009L
45. G. Melfe, A. Fonseca, J. P. Fernandes, Helping developers write energy efficient haskell through a data-structure evaluation, in: 2018 IEEE/ACM 6th International Workshop on Green And Sustainable Software (GREENS), IEEE, 2018, pp. 9–15.
46. F. Ribeiro, J. Saraiva, A. Pardo, Java Stream Fusion: Adapting FP Mechanisms for an OO Setting, in: Proceedings of the XXIII Brazilian Symposium on Programming Languages, SBLP 2019, Association for Computing Machinery, New York, NY, USA, 2019, p. 30–37. doi:10.1145/3355378.3355386.
URL https://doi.org/10.1145/3355378.3355386
47. W. L. Mendonça, R. de Almeida, J. Fortes, F. Lopes, D. Marcílio, E. Canedo, F. Lima, J. Saraiva, Understanding the impact of introducing lambda expressions in java programs, Journal of Software Engineering Research and Development 8

(2020) 8:1–8:22. doi:10.5753/jserd.2020.744.
URL https://sol.sbc.org.br/journals/index.php/jserd/article/view/744