INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA INFORMÁTICA




Effects of Applying Energy Efficient Patterns

ANA CATARINA BARBOSA RIBEIRO
Junho de 2023

POLITÉCNICO
DO PORTO

ISEP INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO P.PORTO

Effects of Applying Energy Efficient Patterns

Ana Catarina Barbosa Ribeiro

A dissertation submitted in partial fulfillment of the requirements for the degree of Master of Science, Specialisation Area of Software Engineering

Supervisor: Isabel Azevedo

Porto, June 30 2023

Declaration of Integrity

I hereby declare having conducted this academic work with integrity.

I have not plagiarised or applied any form of undue use of information or falsification of results along the process leading to its elaboration. Therefore, the work presented in this document is original and authored by me, having not previously been used for any other end.

I further declare that I have fully acknowledged the Code of Ethical Conduct of P. PORTO.

ISEP, June 2023

Ana Catarina Barbosa Ribeiro

Abstract

In recent years, there has been an increase in concern about sustainability. Information and communication technologies are a problem in this regard. Smartphones, tablets, and other linked devices have become increasingly important in society. Additionally, a lot of activities are done online, including shopping, banking, meal ordering, music listening, various public service operations, and so forth. The environmental footprint of information and communication technology has increased because of all these new demands and advances. This way, knowing how to create software that is more environmentally friendly is crucial.

Green Software is an emergent discipline that supports the idea that software should be created and utilized to ensure little or no environmental impact. It covers every stage of a software product's lifecycle, including design, usage, and the effects on the economy, society, and environment. Software practices and architecture, hardware and data center layout, electrical markets, and climate change are all considered by this discipline.

International Energy Agency states that the global data center energy use in 2021 was between 220 and 320 TWh, excluding crypto mining, corresponding to around 0.9-1.3% of the final energy demand. Improvements in hardware and cooling systems are holding back the rise in energy consumption but, even with these advancements, demand growth is still contributing to rising energy consumption, which has increased by 10 to 30% over the past few years. This way, improving the hardware components it’s not enough, it is also needed to look at software energy consumption to increase the efficiency.

The main objective of the work documented here is to understand how the maintainability and performance of the software can be affected when adopting programming techniques to reduce energy consumption. Thus, some approaches and tools were identified.

A Java desktop application was modified. The changes improved the maintainability and the performance of the application, but also, as expected, the energy consumption.

Keywords: energy consumption, Java, maintainability, performance.

Resumo

Nos últimos anos, houve um aumento na preocupação com a sustentabilidade. As tecnologias de informação e comunicação são um problema nesse sentido. Smartphones, tablets e outros dispositivos tornaram-se cada vez mais importantes na sociedade. Além disso, muitas atividades são realizadas online, incluindo compras, transações bancárias, pedidos de refeições, audição de música, várias operações de serviço público, etc. A pegada ambiental da tecnologia de informação e comunicação aumentou devido a todas estes novos avanços. Desta forma, saber criar um software mais amigo do ambiente é crucial.

Green Software é uma disciplina emergente que apoia a ideia de que o software deve ser criado e utilizado de forma a garantir pouco ou nenhum impacto ambiental. Abrange todos os passos do ciclo de vida de um produto de software, incluindo design, uso e os efeitos na economia, sociedade e meio ambiente. As práticas e arquitetura do software, o layout do hardware e dos data centers, os mercados elétricos e as mudanças climáticas são elementos considerados por esta disciplina.

International Energy Agency afirma que o uso global de energia dos data centers em 2021 foi entre 220 e 320 TWh, excluindo mineração de criptomoedas, correspondendo a cerca de 0,9-1,3% do consumo final de energia. Melhorias nos sistemas de hardware e refrigeração têm travado o aumento do consumo de energia, mas, mesmo com esses avanços, o crescimento do uso de software ainda contribui para o aumento do consumo de energia, que aumentou de 10 a 30% nos últimos anos. Assim sendo, as melhorias nos sistemas de hardware não são suficientes, é também necessário analisar o consumo energético do software, de forma a aumentar a sua eficiência.

O principal objetivo deste estudo é entender como a manutenibilidade e a performance do software podem ser afetados ao adotar técnicas para reduzir seu consumo de energia. Assim sendo, foram identificadas algumas técnicas e ferramentas.

Uma aplicação desktop escrita em Java foi modificada. As alterações melhoraram a manutenibilidade e a performance da aplicação, e também, como esperado, o consumo energético.

Palavras-chave: consumo energético, Java, manutenibilidade, performance.

Acknowledgment

I would like to thank all those who were, directly and indirectly, involved in this whole process because this thesis would not have been possible without the cooperation of all these people.

A special thanks to my supervisor, Isabel Azevedo, for all the knowledge and availability. I would like to thank also for all the support and motivation transmitted during this process.

I also want to express my gratitude to my family and friends who have always been there for me and who have helped me through this challenging time.

In closing, I want to express my gratitude to all the professors and colleagues who supported me throughout my academic path at ISEP.

Table of Contents

List of Figures

List of Tables

List of Code Snippets

Abbreviations

AHP	Analytic Hierarchy Process
CI	Consistency Index
CPU	Central Processing Unit
CR	Consistency Ratio
GHz	Gigahertz
GQM	Goal, Question, Metric
ICT	Information and Communication Technology
J	Joule
k	Thousands
kWh	Kilowatt hour
ms	Milliseconds
PUE	Power Usage Efficiency
TEEC	Tool to Estimate Energy Consumption
TWh	Terawatt-hour
RAPL	Running Average Power Limit
W	Watts
ZB	Zettabyte

1 Introduction

This chapter introduces the dissertation developed during the master's degree in Software Engineering at Instituto Superior de Engenharia do Porto. Firstly, a brief contextualization is presented, as well as the definition of the problem, and the investigation's main objectives. Finally, there is a description of the general structure of the document.

1.1 Context

There has been a growing concern about sustainability in the past few years. This concern extends to Information and Communication Technologies (ICTs). It is important to understand how to build greener software.

Green Software is an emerging discipline that defends that software should be developed and used in a way that guarantees minimal, or no, impact on the environment. It includes all the phases of a software product's lifecycle, including the design, use, and economic, social, and ecological impacts. Green Software takes into consideration software practices and software architecture, hardware and data center design, electricity markets, and climate change. The main aim of this discipline is to generate fewer greenhouse gas emissions and reduce the software's carbon footprint (Principles.Green, 2022; Rosencrance, 2022).

The 2030 Agenda for Sustainable Development was endorsed by all the United Nations countries in 2015. Seventeen goals were defined on this agenda: goal number seven related to energy. The objective is to ensure that everyone has access to reliable, sustainable, and modern energy sources. To do this, various stages were laid out, one of which is to double the rate of progress in energy efficiency on a worldwide scale (United Nations, 2022a; World Health Organization, 2022).

Also, the Paris Agreement aims to keep the global temperature rise below two degrees. Since the ratification of this agreement, many countries have committed to net zero emissions targets,

which means that green gas emissions should be reduced to as close to zero as possible. Since the energy industry was recently accountable for around three-quarters of all worldwide greenhouse gas emissions, it is critical to address the issue of energy efficiency (iea, 2022; United Nations, 2022b).

Additionally, Russia's invasion of Ukraine caused the supplies of Russian gas, needed for heating, industrial processes, and power, have been cut by more than eighty percent. This cut caused an unprecedented crisis in the European energy system. One of the most impactful ways to respond to the energy crisis is energy efficiency and conservation (Zettelmeyer et al., 2022).

As well, the evolution of large language models, such as ChatGPT, can contribute to higher energy consumption. There are estimations that ChatGPT's monthly energy consumption now is in the millions of kWh (Kasper Groes Albin Ludvigsen, 2023).

For all the reasons mentioned above, it's important to understand how to make software that consumes less energy, to make the world greener. In the last years, some research on software energy efficiency has been performed. However, many of those focus on mobile applications because smartphones are used to pervasively accomplish important daily tasks, but they have a limited battery life, which can ruin the user experience (Cruz et al., 2019). This work has a different context, as it does not intend to study mobile applications.

1.2 Problem

Society has become increasingly dependent on smartphones, tablets, and connected devices. Also, many tasks are carried out online: listening to music, shopping, bank activities, ordering food, some actions in public services, and so on. All these new needs and developments have led to an increase in the environmental footprint of Information and Communication Technology (ICT).

A report of September 2022 made by the International Energy Agency states that the number of internet users worldwide has more than doubled since 2010, while internet traffic has expanded 20-fold. Global data center energy use in 2021 was between 220 and 320 *TWh*, excluding crypto mining, corresponding to around 0.9-1.3% of the final energy demand. The energy consumed for cryptocurrency is estimated to be between 100 and 140 *TWh* in 2021. Since 2010, the energy consumed by data centers, when excluding crypto mining, has grown only moderately despite the strong growth in demand. This happened due to efficiency improvements in hardware and cooling systems, and a shift away from small and inefficient enterprise data centers to more efficient cloud and hyperscale data centers. However, even with those improvements, the growth in demand is still causing growth in energy consumption, which has been growing between 10 to 30% over the past several years (Kamiya, 2022). To improve the energy efficiency of ICT it's important to keep improving the hardware components but it is not enough, since it is still verified a consumption growth. It is also needed to look at software energy consumption to increase the current efficiency.

This way, it is urgent to tackle this subject and understand how the energy consumed by ICT can be reduced. Some companies are aware of this subject and trying to reduce their footprint. In 2021, Microsoft, Accenture, GitHub, and ThoughtWorks established with the Joint Development Foundation Projects LLC and the Linux Foundation the Green Software

Foundation. This foundation aims to reduce the carbon emissions of software, caused also by energy consumption, encouraging companies and individuals to join the movement. This foundation aims to establish green software standards, promising to create and publish green patterns, and practices across various computing disciplines (Sandquist, 2021). Currently, according to the official website, there are already 38 member organizations, including Intel, Mastercard, Shell, Vmware, and others, and 802 individuals (Green Software Foundation, 2022).

Although Software Sustainability has been more and more of a concern, it is a relatively new topic that is not yet deeply analyzed. For companies and developers is not clear how to build greener software and the impacts it can have on software maintainability and performance. Companies need to understand this relationship to make informed decisions and take action to conserve energy and create a more sustainable future.

1.3 Objectives

The main objective is to understand how the software's maintainability and performance can be affected when adopting techniques to reduce the energy consumption of software. The energy consumption will also be measured and analyzed to validate the application of the techniques.

This way, the research questions are:

1. How the combined use of energy-efficient techniques impacts performance?

2. How the combined use of energy-efficient techniques impacts maintainability?

Some tasks were defined to fulfill the goal and answer the research questions. First, it will be studied how to measure or estimate the energy consumption of software. For that, a search and comparison of the existing tools to measure energy consumption will be performed.

Next, the existent patterns and approaches to decrease software's energy consumption at the source code level will be studied. An application should be used to perform some experiments. Due to the time, it was prioritized the use of an already known application. This way, the experiments will be realized in a desktop application developed in Java.

Finally, based on the original and final solutions, it will be verified that there was a decrease in energy consumption, and it will be analyzed the effects on maintainability and performance quality attributes.

1.4 Document Structure

This document is divided into seven chapters, these being Introduction, State of Art, Value Analysis, Design, Implementation, Tests and Solution Evaluation, and, finally, Conclusions.

The current chapter is the Introduction, whose objective is to present the context, the problem to analyze, and the objectives to achieve. This chapter ends with the current section, explaining the document structure.

The State of Art chapter is divided into five sub-chapters. The first one aims to explain the difference between energy and power consumption. After, a list of tools and methods to measure or estimate the energy consumption of software is presented. Next, there are presented some works where some techniques to reduce energy consumption were explored and their respective conclusions. Lastly, there are presented the quality attributes that will be analyzed.

Next, the Value Analysis chapter includes a value analysis of the research, applying the Analytic Hierarchy Process.

The Design chapter includes a description of all the decisions taken to carry out the controlled experiment. This includes the selection of the tool to measure the energy consumption and the project to use in the experiments.

The Implementation chapter explains the changes performed to the initial application until reaching the final solution.

Then, the Tests and Solution Evaluation chapter define the hypothesis and objective of the evaluation. The evaluation technique to be used is described and explained in this chapter. The results are then presented, followed by an explanation of how they were analyzed to understand the differences between both solutions.

Finally, the Conclusions chapter summarizes what was done, threats to validity, future work, and contributions.

2 State of the Art

This chapter is divided into five sub-chapters. The first sub-chapter defines the difference between energy and power consumption. Next, a list of the available tools to measure or estimate the energy consumption of software is presented. Then, a list of tools to help developers identify energy leaks is shown. After, there are presented some works where some techniques to reduce energy consumption were explored and their respective conclusions. The quality attributes to analyze are presented at the end of the chapter.

2.1 Energy vs Power Consumption

As expressed in section 1.1, Green Software is an emerging discipline that aggregated eight principles that software engineers should follow. One of those principles is electricity, declaring that software should be energy efficient, to reduce carbon emissions.

To improve the energy efficiency of software two measures can be considered: energy and power consumption. They are related, but power represents the amount of energy transferred per unit of time and it's typically measured in Watts (*W*), while energy represents the total energy consumed in a time frame and it's typically measured in Joule (*J*). This way, the energy can be calculated through the formula: $E = P\Delta t$, where P represents the power, in *W*, and Δt represents the amount of time, in seconds (Vedantu, 2022).

Both energy and power consumption have been used to verify if the software's consumption decreased or not, as can be seen in (Pereira et al., 2016) and (Acar et al., 2016), respectively. The value of energy permits verifying if the cumulative value of executing has reduced. The power consumption values allow the verification of the consumption over time, so it's possible to verify peaks and if those peaks decrease.

2.2 Measuring Software Energy Consumption

There are two approaches to support the measurement of software's energy consumption: the hardware-based approach and the software-based approach.

2.2.1 Hardware-based Approach

In a hardware-based approach, a power meter is used to measure energy consumption. Normally, they are digital measurement devices that assess the energy or power consumed directly from the power supply in Joules, or Watts, respectively.

To set up these kinds of tools, they should be connected directly to the socket, and then the computer or server is connected to the tool. Then, to have a reliable value of energy consumption, the consumption of the computer or server should be measured before and during the software's execution. Software energy consumption is thus calculated by subtracting the energy the computer was consuming initially from the energy the computer consumed when executing the software.

This approach is known to be very precise since it allows a direct measurement, instead of just an estimation, but it is not fine-grained and, possibly, represents an additional financial cost. Also, if the software to evaluate is running in the cloud, with no access to the physical server, this approach cannot be used.

First, WattsUp Pro is a device that has been used in some works, for example (Acar et al., 2016). This device can measure nineteen variables, and between those are the real-time power consumption and the cumulative energy consumption. The data collected can be viewed on the device screen but can also be exported via USB to a spreadsheet. This product also works with Logger Pro and LabQuest apps to create graphs (Hirst et al., 2013). WattsUp Pro has been identified as very precise, but it was discontinued, so now the only way to acquire this tool would be second-hand.

Moonson is also a commonly used and precise power meter. This power meter is on sale for 939$ on the Monsoon Solutions Inc. official website (Monsoon Solutions, 2022). This device allows the measurement of energy consumed by mobile devices, so is commonly used to study the power measurement of mobile applications. Also, this tool allows connecting to a computer to obtain the data, including charts (Ghaleb, 2019).

Besides these tools, there are cheaper power meters available in the most common online shops, but normally those are not developed to measure the energy consumption of software, they are developed to measure the energy consumption of home devices, like televisions, fridges, etc. Those devices can be used to measure power consumption, but the precision is not guaranteed, and normally they do not provide the option to export data, they just present values on a screen, which makes the measurement harder and possibly less accurate.

Table 1 represents the comparison between the hardware-based approaches mentioned above.

Table 1 Hardware-based approaches comparison

	WattsUp Pro	**Moonson**	**Cheaper power meters**
Availability	Not available	Available	Available
Cost	Not available	939$	From 20$ to 100$ average on (Amazon, 2023)
Energy Measurement	Entire machine	Entire machine	Entire machine
Precision	Precise	Precise	No guarantee of precision
Data treatment	Yes	Yes	No
Applications	All kind	Android	All kind

2.2.2 Software-based Approach

In this approach, a software tool, normally cost-free, is used to estimate energy consumption. These tools, unlike hardware-based approaches, do not measure the energy, they provide an estimation.

Normally, these software tools base their estimations on other hardware tools. All the tools listed below base the estimations on the Running Average Power Limit (RAPL). Intel CPUs use dedicated RAPL registries to retrieve the power consumption every millisecond, being capable of delivering the energy consumption of the CPU package and DRAM. RAPL tools were studied and (Khan et al., 2018) showed they have a high accuracy.

Intel Power Gadget is a monitoring tool enabled for Intel Core processors from the 2nd up to the 10th generation of processors. This tool is supported both on Windows and macOS operational systems and allows estimations on the energy consumption of the CPU, in Joules. The tool can be used through the command line, but it also provides an interface with some graphics, including CPU utilization and frequency, GPU utilization, GT frequency, and energy consumption. After the log is stopped, an excel file is generated with the elapsed time, CPU frequency, CPU temperature, and average and cumulative energy consumption of the CPU (Intel, 2022).

Powerstat is a command-line software tool that works only in Linux environments. It generates a report with the estimation of the power consumption of the CPU (King, 2015).

PowerAPI is a project that englobes some tools to estimate the energy consumption of devices (RAPL formula), processes (SmartWatts formula), or an entire program (Jouleit). All these tools can be installed and run through the command line. The results will include an estimation of the total energy consumed by the CPU. The tools from PowerAPI only work in Linux operational systems (Spirals research group, 2022).

PowerJoular is a tool released in 2022 that also estimates the CPU's power consumption. The difference (increase or decrease) in power consumption from the last metric will also be shown. The tool is now only available for Linux (Noureddine, 2022).

In addition to these tools, there were more in the past that are not available or were discontinued: Joulemeter (Microsoft, 2010), GreenTracker (Mynatt et al., 2010), Jolinar (Noureddine et al., 2016), and Jalen (Noureddine et al., 2015).

All the tools mentioned above are available online and are free to use. Table 2 compares them.

Table 2 Software-based approaches comparison

	Intel Power Gadget	**Powerstat**	**PowerAPI**	**PowerJoular**
Availability	Available	Available	Available	Available
Cost	Free	Free	Free	Free
Energy Measurement	CPU	CPU	CPU	CPU
Data Treatment	Yes	No	No	No
Environment	Windows macOS	Linux	Linux	Linux

2.2.3 Summary

In this section, a comparison between software-based and hardware-based approaches is performed.

Regarding the reliability of the measurements, the hardware-based approaches are known to be more precise, since they measure the energy supplied by the socket. However, as stated in section 2.2.2, the RAPL method was proven to be accurate, even if it is just an estimation.

The tools that are simple to use are the ones that provide the total amount of energy consumed and allow some data treatment. From the tools analyzed, the ones that allow the generation of excel files or charts are WattsUp Pro, Moonson, and Intel Power Gadget.

Regarding the value of energy that the tools provide, the hardware-based tools will measure the total energy supplied by the socket, so all the energy that the computer is using is considered. The software-based tools will only consider the energy spent by the CPU.

Finally, another aspect to consider is the cost. The hardware-based tools require a financial cost, to obtain the hardware. However, software-based tools are free to use and are available for download online.

2.3 Tools to Identify Energy Leaks

In this section, a list of tools to help developers identify the fragments of code that consume more energy is presented. The main aim of these tools is not to provide values for energy consumption, it is the identification of energy leaks in the code.

2.3.1 E-Debitum

The concept of energy debt represents the additional energy cost of a system caused over time due to the occurrence of energy code smells. Energy code smells correspond to source code approaches that can cause more energy consumption.

In (Maia et al., 2020) the authors based the definition of energy code smells on a review of the state of the art on green software that allows identifying experiments where some approaches were changed and resulted in lower energy consumption and then performed an estimation of the energy cost per usage time, for each one of the smells. After, they express energy debt as a function that considers the number of smells, the context where they were detected, and the expected usage time of the application.

For future work, they defined the construction of an extension to be used within SonarQube, to help in the identification of the energy code smells. Now, this extension is already available – it's called E-Debitum - and can help developers identify source code approaches that can be causing more energy consumption.

This extension has only been tested in Android applications since most of the energy code smells defined are related to Android programming. Also, the authors emphasize that their energy debt approach can be used to have guidelines for energy consumption, but it should not be used to calculate extremely accurate estimates of energy values (Couto et al., 2020).

2.3.2 SPELLing out energy leaks

(Pereira et al., 2020) proposed a technique to detect energy-inefficient fragments in the source code of a software system. To measure the energy consumption test cases are run. Then, to link the energy consumption to the source code, a statistical method, based on spectrum-based fault location, is performed.

The result is a ranking of components sorted by their likelihood of being responsible for energy leaks, essentially pointing the developer's attention to the most critical *red spots* in the analyzed code.

Currently, this technique is available for desktop and server-based systems.

2.4 Reducing Software Energy Consumption

In the last few years, software energy consumption has become an important topic and the subject of many studies. The goal of this chapter is to present some of the studies carried out, to obtain the set of approaches, at the code level, that can be used to reduce the energy consumption of software.

2.4.1 Energy Consumption of Java Collection Framework

In 2016, an analysis of the energy consumption of the Java Collection Framework was done. In this analysis, the methods of three different groups of data structures were considered: *Set*, *List*, and *Map*, and, for each one of these groups, different data sizes were considered (Pereira et al., 2016).

	Concurrent SkipListSet		HashSet		Linked HashSet		TreeSet	
Methods	J	ms	J	ms	J	ms	J	ms
add	1.6822	87	1.7749	87	1.4917	75	1.4817	92
addAll	1.4549	93	1.4771	89	1.9335	94	1.5101	93
clear	1.4901	78	1.0586	64	1.3288	60	1.8566	73
contains	1.4213	88	2.0685	78	1.0401	76	2.0446	79
containsAll	1.8317	96	1.4000	77	2.1748	88	1.4443	89
iterateAll	1.9225	99	1.4554	92	1.2907	83	1.3844	83
iterator	1.6096	83	1.7596	75	0.9613	76	1.7239	76
remove	1.7877	78	1.2633	75	1.2458	93	1.0700	76
removeAll	1.8072	85	2.1359	77	1.9145	100	1.3920	91
retainAll	3.2607	206	2.4092	200	2.2512	199	3.2222	193
toArray	1.4789	86	1.3833	80	1.3776	79	1.6292	80

Figure 1 Set results for a population of 25*k*
From (Pereira et al., 2016)

	ArrayList		AttributeList		CopyOn Write ArrayList		LinkedList		RoleList		Role Unresolved List		Stack		Vector	
Methods	J	ms	J	ms	J	ms	J	ms	J	ms	J	ms	J	ms	J	ms
add	0.9773	71	1.1510	67	1.7839	117	1.8016	86	1.4801	76	1.1865	74	1.5659	76	1.5177	69
addAll	1.3353	76	1.0492	88	1.3586	82	1.1043	88	1.6661	76	1.8672	88	1.1015	88	1.7903	73
addAlll	1.7855	86	1.6035	68	1.1789	86	1.7272	99	1.5980	81	1.2497	85	1.2962	72	1.6268	90
addI	1.7125	93	1.3849	87	1.6558	119	1.6404	96	1.2718	85	1.3124	86	1.5287	83	1.4554	86
clear	1.1284	76	1.2409	75	1.7155	68	1.6497	74	1.6705	76	1.4304	80	1.6199	73	1.0574	71
contains	2.7568	166	2.4228	165	3.1768	167	3.1552	193	2.1751	162	2.4688	164	2.0128	166	2.1558	168
containsAll	1.5993	87	1.8053	92	2.1889	92	2.2887	118	1.3244	100	1.3930	96	1.2054	89	1.5091	87
get	2.0029	83	1.1171	78	1.4918	77	2.0168	109	2.2110	81	1.6613	71	1.8956	86	1.4978	73
indexOf	1.4447	76	2.0325	84	1.5682	70	2.6289	101	1.5674	79	1.1944	81	1.8090	81	2.0788	75
iterateAll	2.0701	79	1.0473	77	1.0103	73	2.6401	107	1.3605	85	1.7822	71	1.6036	81	1.1336	87
iterator	1.4893	84	1.1589	84	1.3922	72	1.7666	108	1.9760	73	1.3300	79	2.1895	84	1.6505	83
lastIndexOf	1.7750	99	1.7666	98	2.0383	94	2.5019	127	1.8914	92	1.4211	95	1.2260	84	1.2296	96
listIterator	1.4457	76	1.6190	84	1.3737	71	2.5003	106	1.3380	80	1.5176	85	1.6354	69	1.2746	81
listIteratori	1.7356	78	1.1552	81	1.5160	77	2.1996	105	1.7588	79	1.0334	80	1.8799	85	1.7545	78
remove	1.1308	96	1.4480	85	2.1946	162	1.6924	98	1.4560	84	1.1368	85	1.2663	96	1.4973	82
removeAll	8.0905	671	7.8108	697	7.3237	666	8.3150	752	7.6148	692	7.9911	664	7.3824	654	7.1281	665
removei	1.9135	85	1.3534	92	2.2858	118	1.7174	100	1.6308	85	1.6369	89	1.5850	81	1.5486	90
retainAll	2.7037	193	2.7845	200	2.6052	198	2.5982	205	3.0973	197	2.4172	200	2.7635	242	3.4019	245
set	0.9476	64	1.5943	70	1.9669	110	2.0474	112	1.5249	76	1.2312	73	1.4938	75	1.4957	72
sublist	1.3108	76	1.6021	80	1.4792	80	1.8457	98	1.4910	85	1.5117	71	1.7082	75	0.9414	75
toArray	1.6418	84	1.5024	84	2.0934	73	1.6739	106	1.5418	79	1.7455	83	1.5694	69	2.0213	80

Figure 2 List results for a population of 25*k*
From (Pereira et al., 2016)

	Concurrent HashMap		Concurrent SkipListMap		HashMap		Hashtable		Linked HashMap		Properties		Simple Bindings		TreeMap		UIDefaults		Weak HashMap	
Methods	J	ms	J	ms	J	ms	J	ms	J	ms	J	ms	J	ms	J	ms	J	ms	J	ms
clear	2.0276	94	2.2961	88	1.8395	104	1.5761	94	1.5025	97	2.0777	98	2.1401	106	1.6706	98	1.8143	105	1.9941	95
containsKey	2.3132	105	2.1693	123	2.1343	103	1.8582	94	1.8726	103	1.6018	107	1.8055	99	1.9452	100	2.3366	89	1.9675	108
containsValue	21.5611	2305	7.8032	643	8.3615	683	8.4957	765	6.1326	462	7.3755	692	7.9912	678	9.1771	847	7.9341	714	6.7072	562
entrySet	2.2878	93	2.2363	116	1.8531	108	2.1332	107	1.8362	113	1.7800	97	2.1557	102	2.1617	115	1.7087	105	1.4666	102
get	2.3106	103	1.9972	119	1.8120	102	1.4071	100	1.8252	116	1.7851	97	1.5359	100	2.2331	115	1.5252	89	1.7185	103
iterateAll	2.1041	96	1.8353	115	2.6673	100	1.5343	91	1.6462	111	1.6362	100	2.0472	116	1.9122	111	1.6574	95	1.7139	106
keySet	1.7287	95	2.4889	124	1.6813	114	2.2226	99	1.8328	103	1.4866	92	2.0630	106	2.1680	110	1.5547	99	1.8749	105
put	1.8591	104	2.2888	102	2.4628	92	1.3123	96	2.0338	108	1.7038	107	2.1646	102	1.4355	91	2.1204	93	2.5784	105
putAll	1.4147	95	2.2852	122	1.7564	100	1.5949	105	1.8608	113	1.3097	95	2.1461	112	1.8914	116	2.3094	87	2.0750	108
remove	1.8574	92	2.2131	105	1.9256	109	1.6067	97	2.2300	106	1.9660	98	2.2178	106	1.8133	101	1.6888	92	2.4103	103
values	1.8279	85	2.4690	116	2.5755	109	2.2266	94	2.0009	107	1.9120	111	2.0692	108	1.4467	105	1.6533	100	2.4628	111

Figure 3 Map results for a population of 25*k*
From (Pereira et al., 2016)

Figure 1, Figure 2, and Figure 3 present each one of the methods analyzed for the *Set*, *List*, and *Map* interfaces, assuming a population of 25 thousand records. For each one of the implementation methods, is presented the energy consumption, in Joules (*J*), and the execution time, in milliseconds (*ms*). The most energy-efficient approach is colored with green, the worst with red, and the others are colored depending on their consumption value when compared to the most inefficient and efficient implementation and colored accordingly.

Considering the values presented above it is possible to get some conclusions (Pereira et al., 2016):

- Figure 1, regarding the *Set* results, permits to verify that the approach that includes most of the more efficient methods is the *LinkedHashSet*. However, this approach is the worst with the *addAll* and *containsAll* methods.

- Regarding the *List* results, presented in Figure 2, it's possible to verify that the *RoleUnresolvedList* and *AttributeList* are the ones that contain the most efficient methods. Although both extend *ArrayList*, they have better consumption values. Also, LinkedList is the worst approach regarding energy consumption.

- Regarding the *Map* approach, presented in Figure 3, we can conclude that the *ConcurrentSkipListMap* is the one that performs poorly. Also, *Hashtable*, *LinkedHashMap*, and *Properties* do not contain any red method.

- Regarding the relationship between energy consumption and execution time, in the add method, comparing *HashSet* and *LinkedHashSet* approaches, we can verify that the execution time and the energy consumption is smaller in *LinkedHashSet*. However, in the clear method, comparing the same approaches, the execution time is smaller for *LinkedHashSet* and this approach is the one that consumes more energy. This way, it is impossible to conclude the correlation between execution time and energy consumption.

The results for populations of 250 thousand and 1 million records can be seen here: https://greenlab.di.uminho.pt/collections/. Analyzing those results, it's possible to conclude that the energy efficiency of the approaches varies according to the size of the data. This way, when choosing a data structure, it's needed to consider the methods most needed and the population size, to choose the best approach.

2.4.2 E-Debitum

In the known energy, code smells identified by the E-Debitum tool (described in section 2.3.1), there are some code smells related to Android programming, but there are others that can apply to programming in general (Maia et al., 2020):

- **Excessive Method Calls:** Calling a method unnecessarily can penalize performance, since a call involves pushing arguments to the stack, storing the return value in the processor's register, and cleaning the stack after. This way calls inside loops should be analyzed to verify if they can be extracted from the loop.

- **Member Ignoring Method:** Static methods are stored in a memory block and, no matter how many class instances are created during the software execution, only one instance of such method will be created and used, which allows less energy consumption. This way, all the methods that can be static should be.

2.4.3 The Impact of Source Code in Software on Power Consumption

In this article, the "Tool to Estimate Energy Consumption" (TEEC) model was presented, promising to help with the estimation of the power consumed by CPU, DRAM, and disk. This model is based on mathematical formulas, to estimate the energy consumption of each one of the elements. Those mathematical formulas were obtained based on the formulas used by the tools presented in section 2.2, and others that are not available anymore, which makes the model just a duplicate of what the tools already provide (Acar et al., 2016).

In the article, some transformations of Java source code were implemented and the energy consumption before and after was calculated through the formulas of the TEEC model and using a hardware device, the WattsUp Pro, to validate the decrease in the energy consumption (Acar et al., 2016).

The transformations performed in the source code were (Acar et al., 2016):

- **Array Copy:** When copying an array to another, it is better to use the *System.arrayCopy()* method than using a loop.

- **Locality of Reference:** Elements close to each other in memory are faster to access. An example of this can be seen in Figure 4. In the unoptimized version, the loop reads the values of 100 elements in an array, and in the optimized version, the loop loads 100 elements, but they are spaced 100 elements apart from each other.

Unoptimized	Optimized

for (int i = 0; i < 1000000; i++) {
int sum = 0;
for (int x = 0; x < 50000; x += 100) {
sum += values[x];
}
}
	for (int i = 0; i < 1000000; i++) {
	int sum = 0;
	for (int x = 0; x < 500; x++) {
	sum += values[x];
	}
	}

Figure 4 Example of Locality of Reference transformation
From (Acar et al., 2016)

- **Array and ArrayList:** When possible, *ArrayList* should be replaced by *Array* since they have a speed advantage.

- **Char Array and StringBuilder:** When possible, *StringBuilder* should be replaced by an array of char data types.

- **Binary Search:** When looking for a value in an array of integer data types, the *Arrays.binarySearch()* method should be used instead of a for loop.

After these transformations were applied to the selected source code, the researchers decreased the software's energy consumption.

2.4.4 Energy Efficiency Analysis of Code Refactoring Techniques for Green and Sustainable Software in Portable Devices

In a recent study, some researchers analyzed the impact of some code refactoring techniques on software energy consumption. For the study, desktop and mobile software were considered but, we will focus only on the desktop results. The source code used to apply the transformations was a C# application of the 2048 game. To estimate the energy consumption on desktop devices, the Intel Power Gadget tool was used (Şanlıalp et al., 2022).

The refactoring techniques analyzed were (Şanlıalp et al., 2022):

- **Encapsulate Field:** This technique consists in setting the access permissions of a variable, making the public fields private, and creating getter and setter methods. This technique can improve software maintainability, but it can worsen software performance since more function calls will be added to the code.

- **Inline Temp:** This technique consists in replacing the variable references by itself, meaning that when we are assigning the return value of a function to a variable that will only be used once, we can replace that variable with the function call and that will save energy because it reduces the access to main and cache memories. This approach also improves performance.

- **Introduce Explaining Variable:** This technique consists in adding variables to the code where there is a complex expression, to make the expression simpler.

- **Simplify Nested Loop:** This technique consists in converting nested loops into a single loop, when possible.

- **Replace Delegation with Inheritance:** Delegation allows the dynamic change of the delegator, which cannot be done with inheritance, in most object-oriented programming languages. But this replacement can reduce the number of lines of code.

The results of this analysis show that all techniques contributed to energy savings. However, the technique that originated better energy consumption values was the "Simplify Nested Loop". The one that saved less energy was the "Replace Delegation with Inheritance".

2.4.5 Energy-Efficient Design Patterns

(Alders, 2016) analyzed the impact that the design patterns State, Template Method, and Strategy had on the energy consumption of software. The energy consumption of these design patterns was compared to non-pattern alternatives.

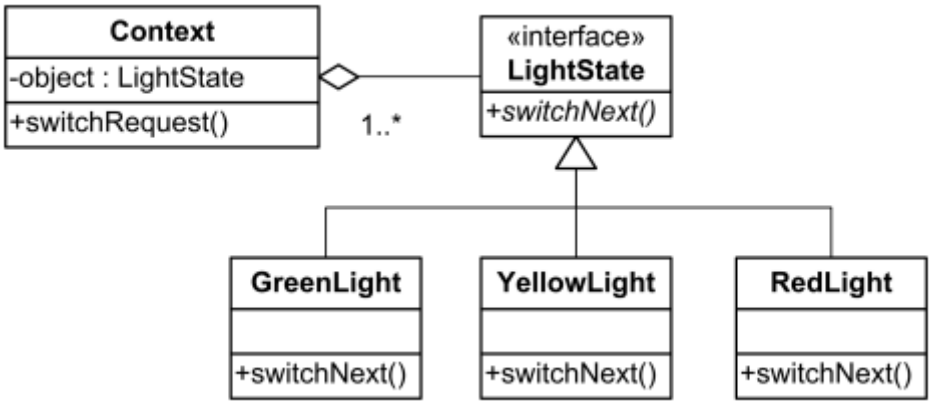

Figure 5 Example of the State pattern
From (Alders, 2016)

The State design pattern allows an object to change its behavior by switching from one state to another. An example of this design pattern is traffic lights, that turn from green to yellow, from yellow to red, and from red back to green. In Figure 5, the traffic light is represented by the *Context* class and the light colors are all the possible state classes. This way, each state is capable to implement behavior particular to itself. The individual states normally are used as singleton objects, and every time the state changes, the current object of the state is replaced with an object of the next state (Alders, 2016).

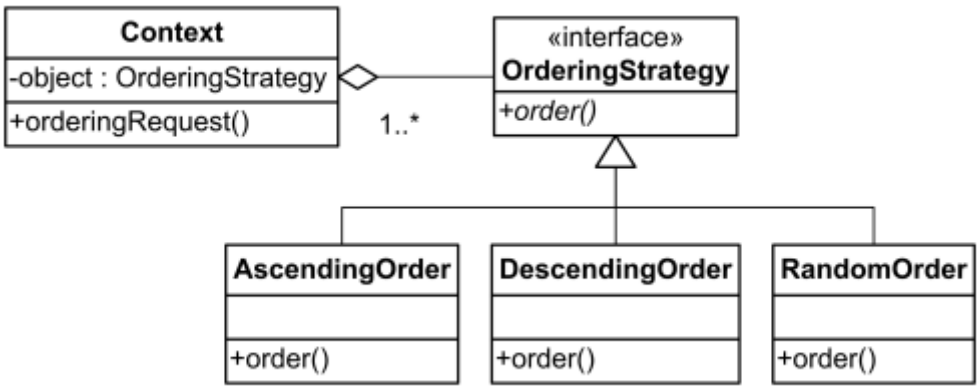

Figure 6 Example of the Strategy pattern
From (Alders, 2016)

The Strategy design pattern allows the encapsulation of algorithms, making them interchangeable. It allows the switch from the currently used algorithm to another different algorithm, depending on the situation. Figure 6 shows an example, considering ordering methods. Depending on the *OrderingStrategy* defined in the Context, a different order method will be run (Alders, 2016).

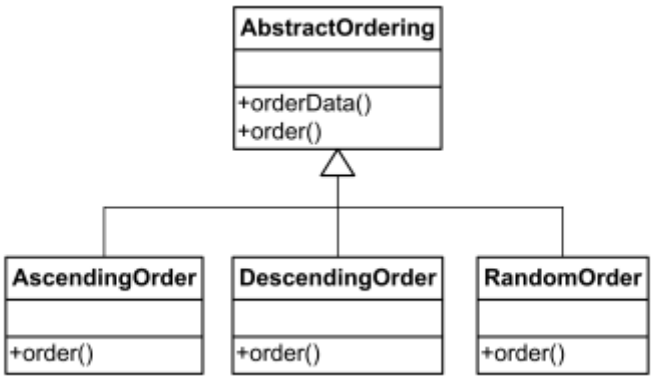

Figure 7 An example of the Template Method
From (Alders, 2016)

The Template Method, as the Strategy pattern, allows the isolation of different algorithms or operations to their subclasses. An adaptation of the Strategy pattern can be observed in Figure 7. The main difference is that Template Method allows the definition of the skeleton of an algorithm in the abstract class and lets subclasses override the steps without changing the overall structure (Alders, 2016).

As said before, the design patterns considered above were replaced by non-pattern alternatives. To select the non-pattern alternatives, the authors explored the existent alternatives for each one of the design patterns, resulting in the following approaches: Replace Conditional with Polymorphism and Form Template Method (Alders, 2016).

The Replace Polymorphism with Conditional consists in removing the polymorphism and keeping all the logic in the Context class. Then conditional statements are used to verify which logic is supposed to be called. Code Snippet 1 shows the result after this transformation, considering the examples presented before (Alders, 2016).

```
public class SortableArray{
    public enum Sorting{
        DescendingOrder,
        AscendingOrder,
        RandomOrder
    };

    private enum currentStateStrategy;

    public int[] sort(int[] list){
        switch(currentStateStrategy){
            case AscendingOrder:
                // Sort in ascending order specific code.
            break;
            case DescendingOrder:
                // Sort in descending order specific code.
            break;
            case RandomOrder:
                // Sort in random order specific code.
            break;
            case default:
                return 0;
            break;
        }
    }
}
```

Code Snippet 1 Code example of the approach Replace Polymorphism with Conditional From (Alders, 2016)

Generally, the Form Template Method transforms the non-pattern code into a Template Method pattern, by separating duplicate code into separate methods. This way, since the goal was to obtain a non-pattern alternative, the process is reversed, resulting in Code Snippet 2.

```java
public abstract class AbstractOrdering{
    public abstract Data[] orderData(Data[] arrayOfData);
}

public class AscendingOrder extends AbstractOrdering{
    @Override
    public Data[] orderData(Data[] arrayOfData){
        // Primitive operation code for preparing the array.
        // Sort in ascending order specific code.
    }
}

public class DescendingOrder extends AbstractOrdering{
    @Override
    public Data[] orderData(Data[] arrayOfData){
        // Primitive operation code for preparing the array.
        // Sort in descending order specific code.
    }
}

public class RandomOrder extends AbstractOrdering{
    @Override
    public Data[] orderData(Data[] arrayOfData){
        // Primitive operation code for preparing the array.
        // Sort in random order specific code.
    }
}
```

Code Snippet 2 Code example for the reversed Form Template Method
From (Alders, 2016)

After the analysis, a general decrease in energy consumption was measured by replacing each design pattern with a non-pattern alternative. The results obtained through this research were the same as expected from the authors. The expectation from the authors was built upon the idea that the non-pattern approaches can be seen as simplifications of the design patterns, so since the code is simpler it should consume less (Alders, 2016).

2.4.6 Empirical Evaluation of the Energy Impact of Refactoring Code Smells

(Verdecchia et al., 2018) analyzed the impact on performance and energy consumption of refactoring well-known code smells on three different Java software applications. The code smells analyzed were:

- **Feature Envy:** this happens when a method is placed in the wrong class. In this case, the method should be placed in the right place.

- **Type Checking:** occurs in *switch* statements and *if/else if* blocks, where a so-called *typefield* is checked for different values, and the according branch is then executed. In this case, the *typefields* should be replaced with a state class.

- **Long Method:** occurs when methods are too large, negatively impacting the readability and maintainability. The solution is to refactor the single method into smaller ones.

- **God Class:** this is a class that controls a significant part of the architecture and implements a large share of the application logic. These classes should be refactored into smaller classes.

- **Duplicated Code:** refers to the duplication of the same code structure in more than one place of the code. In this case, the refactoring is usually carried out by unifying the separate structures into one.

The result of the analysis presented above showed that in one of the three applications analyzed, the refactoring of the code smells caused a significant impact on power and energy consumption.

2.5 Quality Attributes

As explained in section 1.3, one of the objectives of this project is to analyze the impact that the techniques to reduce the energy consumption of software can have on the quality attributes of the software. Therefore, in this section are presented the quality attributes chosen to evaluate the solution, considering that those are the ones that have the most significant impact.

The ISO/IEC 25000 series of standards, also known as System and Software Quality Requirements and Evaluation, has the goal to create a framework for the evaluation of the quality of software. The ISO 25010 from this series, describes the model, consisting of characteristics of software product quality and software quality in use. ISO25010 comprises eight quality characteristics, each one with sub-characteristics (ISO 25000, 2022).

To choose the quality attributes that could be of greater interest to the project, an analysis of carried out studies was done.

(Naumann et al., 2011) shows that for a software product to be considered sustainable, the process to produce the software should also be. This way it is important to consider the lighting of the offices, the energy for transportation purposes, and so on. This way, the time and energy that a developer can take to interpret and change code, to introduce new functionalities, should also be considered. This is also linked with the maintainability quality attribute since if the software is easily maintainable, it will be possible to consume less effort and energy to change or reuse it.

(Cruz et al., 2019) analyzed the impact that changes to improve energy efficiency have on the maintainability of Android applications. A dataset with 539 energy efficiency-oriented commits was used, and the maintainability was measured for each commit. The result of the analysis showed that the software maintainability decreased in 57% of the commits. So, in general, the changes had a negative impact on this quality attribute.

By the analysis presented above, it's possible to conclude that the maintainability quality attribute is important to software sustainability. This way, and since this relation was not yet deeply analyzed in desktop applications, this will be one of the quality attributes that will be analyzed.

(Verdecchia et al., 2018) conducted an empirical evaluation of the impact that refactoring code smells have on the energy consumption and performance of the software. For the analysis, the authors solved some code smells, as described in section 2.4.6, in three Java software applications. The results show that refactoring only some of the identified code smells led to a 49.9% lower energy consumption and 47,8% of performance improvement. However, when

refactoring all the code smells, the performance decreased by 6.8% and the energy consumption decreased by only 10,7%.

(Pereira et al., 2016) analyzed the energy consumption and time of execution of the Java Collection Framework, as presented in section 2.4.1. In this study was shown that some approaches consumed less energy but took more time. However, in some approaches, both time and energy decreased. This way, it was not possible to conclude the relationship between the time that software takes to perform actions and the energy that the software consumes.

Also, in (Capra et al., 2010) was shown that in some cases energy efficiency cannot be increased simply by improving time performance.

This way, it was considered interesting to also analyze the performance quality attribute, to verify if the results obtained allow any conclusion.

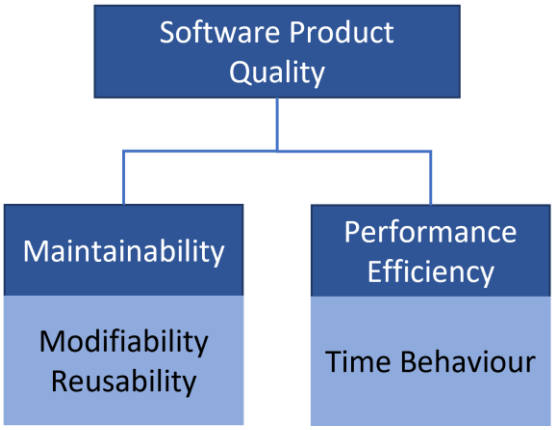

Figure 8 ISO 25010
Adapted from (ISO 25010, 2022)

To summarize, the quality attributes that can be more interesting and important to study are maintainability and performance, so the dissertation will focus on analyzing these two quality attributes. In Figure 8 it's possible to verify the maintainability and performance efficiency quality characteristics of ISO25010, each one with sub-characteristics.

In the following sub-sections, a description of those quality attributes is performed. Also, it is described the sub-characteristics that will be considered.

2.5.1 Maintainability

This characteristic represents the level of efficiency and effectiveness with which the system can be modified to improve, correct, or adapt. The following sub-characteristics were chosen for the analysis (ISO 25010, 2022):

- **Modifiability:** analysis of the level to which the system can be easily modified without introducing defects or degrading the existing product quality.

- **Reusability:** analysis if an asset can be used in more than one system.

2.5.2 Performance Efficiency

This characteristic represents the performance relative to the number of resources used. The following sub-characteristic was chosen for the analysis (ISO 25010, 2022):

- **Time Behaviour:** analysis of the time that the system takes to respond and process when performing its functions.

3 Value Analysis

This chapter represents the project's value analysis, which is a mandatory requirement of the curriculum for the module Value Analysis.

The value analysis of the project was made based on the New Concept Development Model. A description of the model is made and then, in the following sections the elements of the model are defined. Firstly, there are the Opportunity Identification and Opportunity Analysis sections. Additionally, there is a section called Idea Generation, where the alternatives to measure the energy consumption of software are defined. Lastly, it was used the Analytic Hierarchy Process to aid in choosing one of the alternatives, presented in the section Idea Selection.

3.1 New Concept Development Model

The New Concept Development Model consists of three parts, as presented in Figure 9.

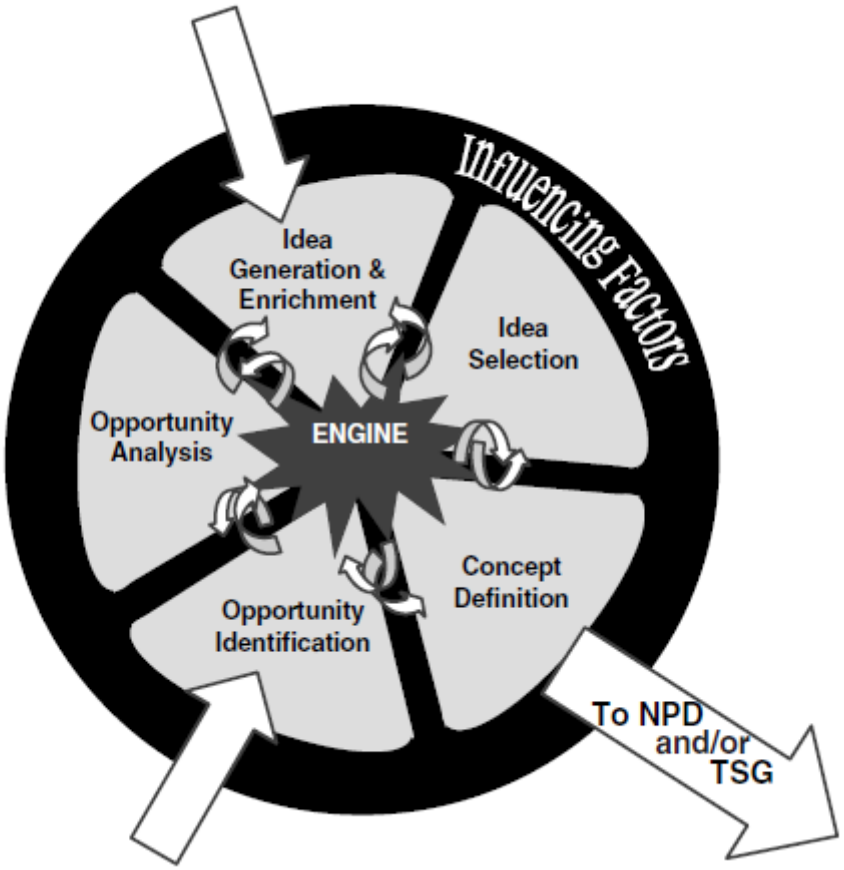

Figure 9 The New Concept Development Model
From (Koen et al., 2002)

The first part, designated by "Engine", represents the organization's characteristics, that drive the five elements, controllable by the organization.

The second part, the inner spoke area, defines the five controllable activity elements:

- **Opportunity Identification:** in this element, the organization identifies opportunities to pursue. These opportunities are defined based on the company's business goals.
- **Opportunity Analysis:** in this element the opportunities identified are analyzed, to verify if they are worth pursuing.
- **Idea Generation:** in this element, a concrete idea is born, developed, and maturated. This process can be formal, such as a brainstorming session, or it can be informal, like a user making an unusual request.
- **Idea Selection:** in this element, the ideas to pursue should be selected. However, there is any process that guarantees a good selection and because of that normally organizations have some difficulties choosing the ideas that allow more business value.

- **Concept Definition:** in this element, the business plans are developed for the selected ideas.

The third and last part is the influencing factors, that can influence the activity elements. These factors can be organizational capabilities, outside world influence, like government policy, customers, competitors, and the enabling sciences that can be involved.

3.1.1 Opportunity identification

In 2015, all countries in the United Nations adopted the 2030 Agenda for Sustainable Development. This agenda defines seventeen goals to be accomplished by 2030 (World Health Organization, 2022). Goal number seven is about energy and intends to guarantee access to reliable, sustainable, and modern energy sources for all. To accomplish this, some steps were defined and one of them is to double the global rate of improvement in energy efficiency, as shown in Figure 10 (United Nations, 2022a).

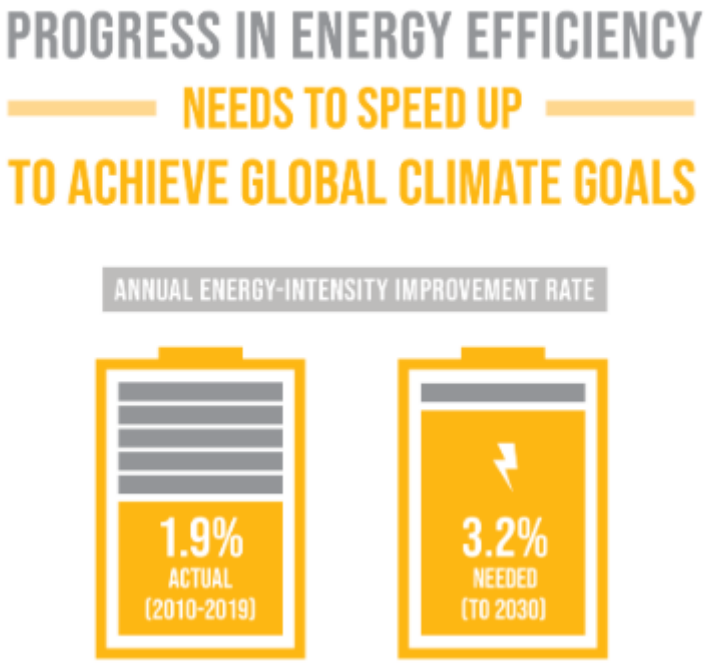

Figure 10 Sustainable Development Goals: Goal 7
From (United Nations, 2022a)

Related to the United Nations Sustainable Development Goals, is the Paris Agreement, which aim is to keep the global temperature rise below two degrees. Since the adoption of this agreement, and to accomplish the goal, a lot of countries have committed to net zero emissions targets, which means that green gas emissions should be reduced to as close to zero as possible. In recent years, the energy sector was responsible for around three-quarters of global

greenhouse gas emissions, so it is urgent to tackle the energy efficiency problem. (iea, 2022; United Nations, 2022b)

Also, these years were marked by Russia's invasion of Ukraine. Caused by that, the supplies of Russian gas, needed for heating, industrial processes, and power, have been cut by more than eighty percent. This cut caused an unprecedented crisis in the European energy system. One of the most impactful ways to respond to the energy crisis is energy efficiency and conservation (Zettelmeyer et al., 2022).

The project to be developed intends to discover how to develop software that consumes less energy and how that can affect the maintainability and performance quality attributes of the software. This project can help in reducing the energy consumption of software and, consequentially, help in the progress of energy efficiency.

3.1.2 Opportunity Analysis

In the last few years, the demand for digital services has grown rapidly. Global data center electricity use, in 2021, was around 0.9%-1.3% of global final electricity demand, excluding the energy used for cryptocurrency mining (Kamiya, 2022).

In Table 3 it's possible to verify the growth of internet usage, and consequentially, the energy usage, from 2015 to 2021.

Table 3 Growth of internet and energy usage
From (Kamiya, 2022)

	2015	**2021**	**Change**
Internet users	3 billion	4.9 billion	+60%
Internet traffic	0.6 *ZB*	3.4 *ZB*	+440%
Data center energy use (excluding crypto)	200 *TWh*	220-320 *TWh*	+10-60%
Crypto mining energy use	4 TWh	100-140 *TWh*	+2 300-3 300%
Data transmission network energy use	220 *TWh*	260-340 *TWh*	+20-60%

Regarding greenhouse gas emissions, the data centers, and data transmission networks are responsible for 0.9% of energy-related greenhouse gas emissions, corresponding to 0.6% of the global emissions (Kamiya, 2022).

As was said before, the demand for data center services has grown strongly however, since 2010 data center energy use (excluding crypto) and greenhouse gas emissions have grown only moderately because of improvements in IT hardware and cooling systems, allowing less energy consumption.

This is good progress, but it still represents growth, and what is needed is a decrease in energy use and the emissions of greenhouse gases. Hardware improvement can, and should, continue but it is also needed to check how to reduce this energy use within the software.

3.1.3 Idea Generation

Through opportunity identification and analysis, it's possible to conclude that the energy consumption of software should be analyzed and reduced. The big question that remains is how to measure the energy consumption of software.

In section 2.2 a list of tools available to measure the energy consumption of software is presented. Since the computer to be used runs on a Windows operating system, and some hardware-based approaches are not accessible, the only options available are the Intel Power Gadget and power meters.

This way, the following alternatives are available:

- **Alternative 1:** Use Intel Power Gadget to measure energy consumption.
- **Alternative 2:** Use a power meter to measure energy consumption.
- **Alternative 3:** Use Intel Power Gadget and a power meter to estimate energy consumption and use both results to perform the analysis.

3.1.4 Idea Selection

To select one of the alternatives presented, the method Analytic Hierarchy Process (AHP) will be used. For that, it is needed to define the decision criteria and the alternatives, already presented in the previous section.

The decision criteria defined were:

- **Simplicity:** the ease with which the energy can be estimated.
- **Reliability:** the known accuracy of the energy estimations.
- **Functionalities:** the number of functionalities that the tool provides that can be interesting to the investigation.

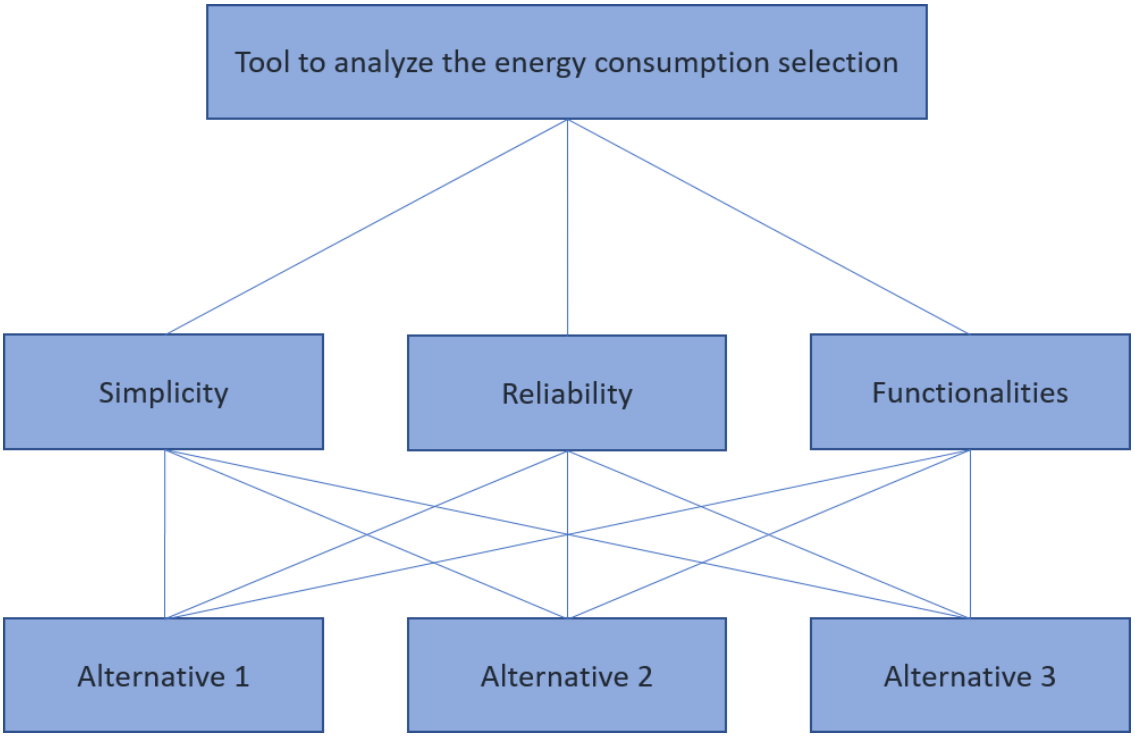

Figure 11 Decision hierarchic tree

The first step is the creation of the decision hierarchic tree, which is represented in Figure 11.

After, it is needed to define the level of importance of each criterion. For that, the scale of Saaty (Saaty, 1990) is used. Mapping this scale with the decision hierarchic tree results in the criteria comparison matrix represented in Table 4.

Table 4 Criteria comparison matrix

	Simplicity	**Reliability**	**Functionalities**
Simplicity	1	1/5	1/4
Reliability	5	1	2
Functionalities	4	1/2	1

Then, the criteria comparison matrix needs to be normalized. To normalize the matrix, each value is divided by its column sum. After the normalization, to obtain the priority vector, the mean of each line is calculated and represents the relative priority of each criterion. The result of these steps is represented in Table 5.

Table 5 Normalized criteria matrix

	Simplicity	**Reliability**	**Functionalities**	**Relative Priority**
Simplicity	1/10	2/17	1/13	0.098
Reliability	1/2	10/17	8/13	0.568
Functionalities	2/5	5/17	4/13	0.334

After, it's needed to evaluate the consistency of the results. To do that it is needed to calculate the consistency index (*CI*) and the consistency ratio (*CR*).

To calculate the consistency index, the following formula is used: $CI = (\lambda_{max} - n) \ / \ (n - 1)$, where n corresponds to the number of criteria, so n=3. The value of λ_{max} is calculated by multiplying the normalized matrix with the priorities vector and then calculating the mean between the result obtained and the values from the priorities vector. By performing those calculations, $\lambda_{max} = 3.025$, resulting in $CI = 0.012$ (Nicola, 2022).

To calculate the consistency ratio, the following formula is used: $CR = CI/0.58$, resulting in $CR = 0.021$. Since *CR* < 0.1, we can conclude that the relative priorities are consistent (Nicola, 2022).

In the next step, are defined the matrixes of comparison for each one of the criteria, considering each one of the alternatives identified. In Table 6, Table 7, and Table 8 it's possible to verify the matrix and the respective priorities vector for each one of the criteria.

Table 6 Comparison matrix and priorities vector for simplicity criteria

Simplicity	**Alternative 1**	**Alternative 2**	**Alternative 3**	**Relative Priority**
Alternative 1	1	3	5	0.648
Alternative 2	1/3	1	2	0.230
Alternative 3	1/5	1/2	1	0.122

Table 7 Comparison matrix and priorities vector for reliability criteria

Reliability	**Alternative 1**	**Alternative 2**	**Alternative 3**	**Relative Priority**
Alternative 1	1	5	1/2	0.380
Alternative 2	1/5	1	1/3	0.118
Alternative 3	2	3	1	0.501

Table 8 Comparison matrix and priorities vector for functionalities criteria

Functionalities	**Alternative 1**	**Alternative 2**	**Alternative 3**	**Relative Priority**
Alternative 1	1	5	1	0.480
Alternative 2	1/5	1	1/3	0.115
Alternative 3	1	3	1	0.405

By analyzing the values presented above, it's possible to conclude that Alternative 1 has the highest relative priority in criteria Simplicity and Functionalities, but in Reliability, the alternative that represents the biggest value is the third one.

Finally, the matrix of the relative vectors for each one of the alternatives, represented in the tables above, is multiplied by the priority vector calculated previously, represented in Table 5, obtaining the global weight for each one of the alternatives:

- Alternative 1: 0.440
- Alternative 2: 0.128
- Alternative 3: 0.432

Through the values presented above, it's possible to conclude that the best alternative is Alternative 1, followed by Alternative 3, and finally Alternative 2.

4 Design

In this chapter, all the decisions taken to carry out the controlled experiment are described. It was needed to select the tool to measure the energy consumption (section 4.1), and the project to use in the experiments (section 4.2).

4.1 Selected Tool to Measure Energy Consumption

A list of the available tools to measure energy consumption was presented in section 2.2. Since WattsUp Pro is not available and Moonson only measures the energy of Android applications, the only option available from the hardware-based approaches is the cheaper power meters.

From the software-based approaches, the only tool that could be used is the Intel Power Gadget since the computer to use in the experiment is an Asus with an Intel Core i7-7500U CPU, running on Windows.

In sections 3.1.3 and 3.1.4 the AHP method was used to aid in choosing between using the power meter, the Intel Power Gadget, or both. The decision criteria defined were simplicity, reliability, and the functionalities available. This analysis showed that the best alternative is the Intel Power Gadget, which is the tool to use.

4.2 Selected Project

As mentioned earlier (section 1.3), an application will be used to perform some experiments. When choosing the application some characteristics were considered. The knowledge of the application and the documentation are important aspects because of the time limitation. Also, it should be an application with a minimum level of complexity, to allow the application of some of the techniques described in 2.4. Finally, the application should have a good maintainability level because if it doesn't any small change in the code will cause a considerable effect on this quality attribute, and that can compromise the results.

This way, the project chosen is a backend service developed within a curricular unit of the master's degree. To use this source code was asked for permission from the teacher Isabel

Azevedo, responsible for the curricular unit where the application was developed and the supervisor of this thesis, and the other two developers: André Alves and José Ferreira. The permissions given by the mentioned people can be consulted in Appendix A.

The selected project is called GourgeousSandwich and is available on GitHub, in the following repository: https://github.com/1170972/GorgeousSandwich.

The chosen project is a monolithic developed in Java language, using Spring Boot framework and Spring Data JPA to connect to the H2 database.

4.2.1 Business Context

In this section, the use cases of the application are presented, providing the necessary information to understand the business context of the system.

Table 9 identifies and describes the use cases of the application. The goal is to apply techniques to reduce energy consumption without changing the functionalities so that it is possible to compare the results of the two solutions later. To guarantee that the use cases were not affected, unit tests will be developed, since the application chosen doesn't have those implemented.

Table 9 Use cases

Identification	Description
UC-01	The user must be able to define a sandwich.
UC-02	The user must be able to update the stock of a sandwich.
UC -03	The user must be able to see the list of sandwiches.
UC -04	The user must be able to see the details of a sandwich.
UC -05	The user must be able to review/rate a sandwich.
UC -06	The user must be able to see the reviews of a sandwich.
UC -07	The user must be able to see all his reviews.
UC -08	The user must be able to know the rate interval (minimum and maximum).
UC -09	The user must be able to comment on a sandwich.
UC -10	The user must be able to see the comments of a sandwich.
UC -11	The user must be able to see all his comments.
UC -12	The user must be able to make an order.
UC -13	The user must be able to update the order.
UC -14	The user must be able to see all orders.
UC -15	The user must be able to see the details of an order.
UC -16	The user must be able to see the possible delivery times.

4.2.2 Architecture

To support the functionalities in this system, a collaboration of components in different layers is required. Figure 12 represents the implementation view of the selected project.

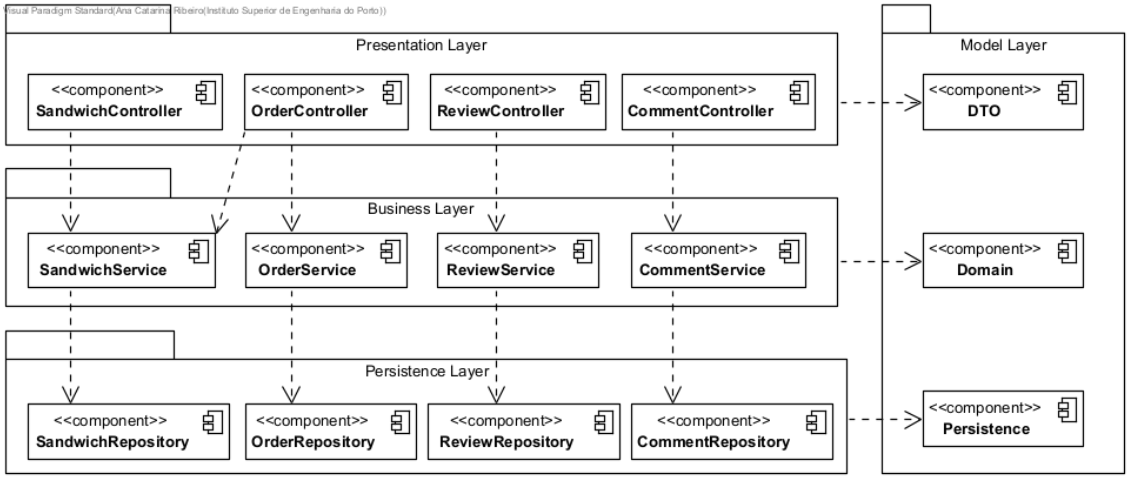

Figure 12 Layer Diagram

- **Presentation Layer:** this layer is responsible for the interface, defining the endpoints, and forwarding requests to the underlying layer, the business layer.
- **Business Layer:** this layer contains the core logic of the system. Data required processes are queried from the layer below, the persistence layer.
- **Persistence Layer:** this layer is responsible to read, store, and updating the data in the database.
- **Model Layer:** this layer encapsulates the data from the communications between the external client and the presentation layer (DTO), the presentation layer and the business layer (Domain), and between the business layer and the persistence layer (Persistence).

Figure 13 represents a process view of the system, having the definition of a sandwich use case as an example.

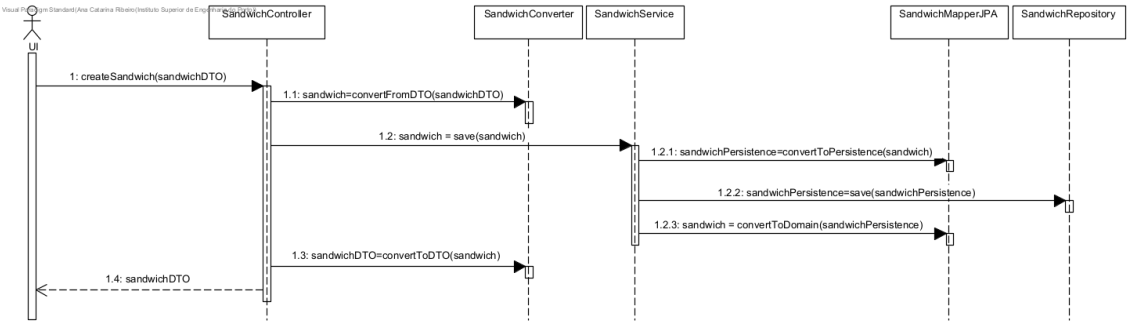

Figure 13 Define a sandwich request example

5 Implementation

The focus of this chapter is to explain the changes performed to the initial application. Initially, a description of the unit tests implemented is provided. Then, it is described and explained the techniques applied to reach the final solution.

The final version of the application can be seen in the same GitHub repository mentioned earlier (section 4.2). The repository has two folders, one for the initial version and another for the final.

5.1 Unit Tests

Unit tests were developed for each one of the controller classes. These tests were developed to ensure the following:

- All the functionalities of the initial solution are available also in the final version.
- Methods inputs are the same in the initial and final solutions.

To guarantee that the unit tests are not dependent on the data that exists in the database, the information was simulated using mocks, through Mockito.

All the test classes have a setup method that runs before the tests, to create some necessary data, as can be seen in Code Snippet 3. After, there is a test method for each one of the endpoints available. An example of the test to the endpoints responsible for creating a sandwich and updating the stock can be seen in Code Snippet 4.

```java
@BeforeClass
public static void setup(){
    DeliveryTime.changePossibleIntervals(
            GorgeousSandwichApplication.calculateIntervals(LocalTime.parse("08:00"),
            LocalTime.parse("22:00"), interval: 20));
    Grade.changeMinMax(1, 5);
    header.setContentType(MediaType.APPLICATION_JSON);

    sandwich = new SandwichDTO();
    sandwich.stock = 1;
    sandwich.type = "salty";
    sandwich.designation = "Test Sandwich 1";
    sandwich.description = "Description of Sandwich 1";

    testSandwich1 = new SandwichDTO();
    testSandwich1.stock = 2;
    testSandwich1.type = "sweet";
    testSandwich1.designation = "Test Sandwich 2";
    testSandwich1.description = "Description of Sandwich 2";

    testSandwich2 = new SandwichDTO();
    testSandwich2.stock = 3;
    testSandwich2.type = "sweet";
    testSandwich2.designation = "Test Sandwich 3";
    testSandwich2.description = "Description of Sandwich 3";

    allsandwich = new ArrayList<>();
    allsandwich.add(sandwich);
    allsandwich.add(testSandwich1);
    allsandwich.add(testSandwich2);
}
```

Code Snippet 3 Example of a setup method

```java
@Test
public void addSandwich() {
    when(sandwichController.createSandwich(sandwich)).thenReturn(sandwich);
    assertEquals(sandwich, sandwichController.createSandwich(sandwich));
}

@Test
public void changeStockSandwich() throws Exception {
    SandwichDTO sandwichUpdated = sandwich;
    sandwichUpdated.stock = 2;
    ResponseEntity<SandwichDTO> responseEntity = new ResponseEntity<>(sandwichUpdated, header, HttpStatus.OK);

    when(sandwichController.addUnitsSandwich(sandwich.sandwichId, sandwichUpdated)).thenReturn(responseEntity);
    assertEquals(sandwichUpdated,
            sandwichController.addUnitsSandwich(sandwich.sandwichId, sandwichUpdated).getBody());
}
```

Code Snippet 4 Example of the developed tests

5.2 Techniques Applied

In this chapter, the modifications done to the application will be explained. The techniques that could contribute to a decrease in the energy consumption of the selected software were listed in section 2.4. Those techniques are subjective to the project, not all the techniques will apply to all projects. This way, the source code was analyzed to find faults where those techniques could be applied. It was identified faults for seven of the techniques analyzed.

Table 10 shows the number of faults encountered and changed, distributed by technique. Appendix B lists the classes or methods where the faults were found. Additionally, all the methods and imports not used were removed.

Table 10 Number of faults distributed by technique

Technique	Number of Faults
Inline Temp + Introduce Explaining Variable	43
Member Ignoring Method	31
Excessive Method Calls	7
Java Collection Framework	5
Feature Envy	3
Encapsulate Field	2

The following sections will describe the implementation of each one of the techniques.

5.2.1 Inline Temp and Introduce Explaining Variable Techniques

These techniques are described in section 2.4.4 and were analyzed together since they are related. The Inline Temp technique consists in replacing the variable reference with the return value of a function if a variable will only be used once. Meanwhile, the Introduce Explaining Variable technique consists in adding variables to the code where there is a complex expression, to make the expression simpler. In the cases where the application of the first one would make the expressions too complex, the second technique applies, and it was not changed.

Forty-three faults were identified and corrected. Code Snippet 5 and Code Snippet 6 show an example of the application of the Inline Temp technique, respectively before and after the alterations.

```
@GetMapping("/sandwiches")
public List<SandwichDTO> listAll() {
    List<Sandwich> sandwiches = sandwichRepository.findAll();
    return sandwichConverter.convertListToDTO(sandwiches);
}
```

Code Snippet 5 listAll() method before changes

```java
@GetMapping("/sandwiches")
public List<SandwichDTO> listAll() {
    return SandwichConverter.convertListToDTO(sandwichRepository.findAll());
}
```

Code Snippet 6 listAll() method after changes

5.2.2 Member Ignoring Method Technique

This technique is described in section 2.4.2 and consists of, when possible, using static methods instead of non-static ones. Thirty-one faults were identified and corrected. The classes where more faults were encountered were those responsible for translating between the dto, domain, and persistence layers. In Table 23 from Appendix B is possible to see the list of methods that were changed.

5.2.3 Excessive Method Calls Technique

This technique is described in section 2.4.2 and it consists in be careful with methods that are called unnecessarily or called too many times unnecessarily. Seven faults were identified and corrected.

In Code Snippet 7 and Code Snippet 8, it's possible to verify an example of a method that is being called unnecessarily. The first image shows the method responsible to update a sandwich stock, at the controller level. The first statement queries a sandwich with that identification, to verify if the identification is valid. After that, and if it is valid, a call to the method presented in Code Snippet 8 is performed. The second method also verifies if the identification of the sandwich is valid. There are two validations of the same thing, so the method is called twice unnecessarily. Considering this, the second validation was removed.

```java
@PutMapping("/sandwiches/{id}")
public ResponseEntity<SandwichDTO> addUnitsSandwich(@PathVariable(value = "id") Long sandwichId,
                                                    @RequestBody SandwichDTO sandwichDTO)
                                                    throws ResourceNotFoundException {
    try {
        Sandwich sandwich = sandwichRepository.getById(sandwichId);
        sandwich.changeStock(sandwichDTO.stock);
        this.sandwichRepository.update(sandwich);
        return ResponseEntity.ok().body(sandwichConverter.convertToDTO(sandwich));
    }catch (NoSuchElementException e){
        throw new ResourceNotFoundException("Sandwich not found with id " + sandwichId);
    }
}
```

Code Snippet 7 Method addUnitsSandwich() of SandwichController

```
public Sandwich update(Sandwich model) {
    if(this.repository.existsById(model.obtainSandwichID().obtainID())) {
        SandwichPersistenceJPA sandwichJPA = this.mapper.convertToPersistence(model);
        return this.mapper.convertToDomain(this.repository.save(sandwichJPA));
    } else {
        throw new NoSuchElementException();
    }
}
```

Code Snippet 8 Method update() from SandwichRepositoryWrapperJPA

5.2.4 Java Collection Framework Technique

This study is described in section 2.4.1 and it allows the developer to understand the best implementation to use, from the Java Collection Framework. This way, it was searched in the code for all the uses of an implementation of the Java Collection Framework. After that, it was analyzed the methods used and if there was another implementation that consumes less energy for that specific method, to always use the implementation more energetically efficiently. In this analysis, it was considered a population of 25k. A total of five faults were encountered.

Code Snippet 9 presents an example of one occurrence that was changed. In this case, a HashSet was being used and the method used is the add(). The HashSet is the implementation that consumes more energy in the add execution. After verification, it was realized that the implementation that consumes less energy in the add() function is the TreeSet. Because of that, the HashSet was replaced by a TreeSet implementation.

```
public Set<OrderItemDTO> convertOrderItemsListToDTO(Set<OrderItem> orderItems){
    Set<OrderItemDTO> orderItemsDTO = new HashSet<>();
    for (OrderItem o : orderItems){
        orderItemsDTO.add(convertOrderItemToDTO(o));
    }
    return orderItemsDTO;
}
```

Code Snippet 9 convertOrderItemsListToDTO() from OrderConverter before changes

5.2.5 Feature Envy Technique

This technique is described in section 2.4.6 and it consists in putting a method in the right class when it is not in the right place. Three faults were identified and corrected. In Code Snippet 10 and Code Snippet 11, it's possible to verify an example of a logic that is being done in the wrong place.

When creating or updating an order, the delivery time selected needs to be one of the available delivery times, calculated based on the definition of the opening hours, closing hours, and interval, defined on a configuration file. In Code Snippet 10 it's possible to see the initialization of the application, where the properties mentioned are read from the configuration file and stored in variables of the DeliveryTime class. Then, every time an order is created or updated or a call to know the possible delivery times is done, the method present in Code Snippet 11

executes. Since the properties read from the configuration file are read at the beginning of the application, instead of calculating the delivery times every time they are needed, they can be calculated only once in the initialization. So that change was done. The method calculateIntervals() was changed to the class that initializes the application and, instead of storing the values from the file in the properties in the DeliveryTime class, the intervals are calculated and what is stored in the class is the list of the available times, as it is possible to verify in Code Snippet 12. This way, every time this data is needed, it's just a return, instead of a new processing.

```
public static void main(String[] args) {
    Properties properties = new Properties();
    try {
         FileInputStream ip = new FileInputStream( name: "./src/main/resources/config.properties");
         properties.load(ip);
         DeliveryTime.OpeningHours = LocalTime.parse(properties.getProperty("openingHours"));
         DeliveryTime.ClosingHours = LocalTime.parse(properties.getProperty("closingHours"));
         DeliveryTime.Interval = Integer.parseInt(properties.getProperty("interval"));
        Grade.MIN_VALUE = Integer.parseInt(properties.getProperty("grade.min"));
        Grade.MAX_VALUE = Integer.parseInt(properties.getProperty("grade.max"));
    }catch (IOException e){
        DeliveryTime.OpeningHours = LocalTime.parse("08:00");
        DeliveryTime.ClosingHours = LocalTime.parse("22:00");
        DeliveryTime.Interval = 20;
        Grade.MIN_VALUE = 1;
        Grade.MAX_VALUE = 5;
    }

    SpringApplication.run(GorgeousSandwichApplication.class, args);
}
```

Code Snippet 10 Initialization of the application before changes

```
public static List<DeliveryTimeDTO> calculateIntervals(){
    List<DeliveryTimeDTO> list = new ArrayList<>();
    LocalTime hours = OpeningHours;
    while(!hours.equals(ClosingHours)){
        LocalTime end = hours.plusMinutes(Interval);
        list.add(new DeliveryTimeDTO(hours.toString(),end.toString()));
        if(end.plusMinutes(Interval).isAfter(ClosingHours)){
            list.get(list.size()-1).endTime=ClosingHours.toString();
            break;
        }
        hours=end;
    }
    return list;
}
```

Code Snippet 11 Method calculateIntervals() in DeliveryTime class

```java
public static void main(String[] args) {
    Properties properties = new Properties();
    try {
         FileInputStream ip = new FileInputStream( name: "./src/main/resources/config.properties");
         properties.load(ip);
         DeliveryTime.changePossibleIntervals(
                 calculateIntervals(LocalTime.parse(properties.getProperty("openingHours")),
                                    LocalTime.parse(properties.getProperty("closingHours")),
                                    Integer.parseInt(properties.getProperty("interval"))));
        Grade.changeMinMax(Integer.parseInt(properties.getProperty("grade.min")),
                           Integer.parseInt(properties.getProperty("grade.max")));
    }catch (IOException e){
        DeliveryTime.changePossibleIntervals(
                calculateIntervals(LocalTime.parse("08:00"), LocalTime.parse("22:00"),  interval: 20));
        Grade.changeMinMax(1, 5);
    }

    SpringApplication.run(GorgeousSandwichApplication.class, args);
}
```

Code Snippet 12 Initialization of the application after changes

5.2.6 Encapsulate Field Technique

This technique is described in section 2.4.4 and it consists in setting the access permissions of a variable, making the public fields private, and creating getter and setter methods. Two faults were identified and corrected. The faults were present in the Grade class, two of the fields were public. This way, they were converted to private, and the method to set the values was created. The values of those attributes are read-only in the class, so it was not needed to implement the getter methods.

6 Tests and Solution Evaluation

This chapter presents the methodology used to analyze and evaluate the impact that applying techniques to reduce the energy consumption of software can have on maintainability and performance. The method Goal, Question, Metric (GQM) was used to define the metrics to evaluate the energy use, performance, and maintainability of the software.

The results are then presented, followed by an explanation of how they were analyzed to understand the differences between both solutions.

6.1 Methodology

The GQM measurement model consists of three levels (Basili et al., 1994):

- **Conceptual Level (Goal):** a goal is defined for an object, relative to a particular environment. Objects of measurement can be products, processes, or resources.
- **Operational Level (Question):** define a set of questions based on the goal. Those questions aim to verify if the objective has been achieved.
- **Quantitative Level (Metric):** for each question, a set of metrics is defined, objective or subjective, to answer the question quantitatively.

The GQM model starts with the definition of a goal. This goal is then refined into several questions, that break down the issue into its major components. Next, each question is refined into metrics. The same metric can be used in different questions, under the same goal. Once the model is defined, the data-collecting strategies to gather data for the metrics must be determined (Basili et al., 1994).

As said before, the GQM method will be used to evaluate the impact that applying techniques to reduce the energy consumption of software can have on the quality attributes and the energy use. This way, the same questions, and metrics will be used to evaluate the project before and after, allowing the comparison of the results obtained.

Table 11 GQM - Goal

Goal	Find out the impacts that applying techniques to reduce energy consumption can have on software's energy use, performance, and maintainability.

The goal defined is presented in Table 11. In the following sections, it will be presented the questions and metrics defined for each element to evaluate.

6.1.1 Energy Consumption

In Table 12 it's possible to verify the questions defined for the energy consumption, and the metric that will be used to evaluate.

Table 12 GQM Energy Consumption

Question	**Metric**
What is the energy consumption of the initial application?	Cumulative Processor Energy - Intel Power Gadget
What is the energy consumption of the modified application?	

The Intel Power Gadget tool can measure the energy consumption of the CPU, in Joules. This tool, described in section 2.2.1, will be used to measure the energy use of the software before and after the changes, to validate that the consumption has decreased.

6.1.2 Performance

Table 13 shows the question defined for the performance quality attribute and the metrics that will be used to evaluate.

Table 13 GQM Performance

Question	**Metrics**
Is the performance of the initial application acceptable?	Total Elapsed Time - JMeter Throughput - JMeter
Is the performance of the modified application acceptable?	

The two metrics to be considered when evaluating the performance will be the Total Elapsed Time and the Throughput, provided by JMeter since it allows these two calculations easily. The Total Elapsed Time is calculated in seconds and represents the time passed during the run of the Test Plan, so the lower this value, the better. The Throughput represents the number of requests the application can serve per second. This way, an application is better in performance if its throughput is higher.

6.1.3 Maintainability

In Table 12 it's possible to verify the question defined for the maintainability quality attribute and the metrics that will be used to evaluate.

Table 14 GQM Maintainability

Question	Metrics
Is the initial application easily maintainable?	Maintainability Rating - SonarQube Maintainability Level - SonarGraph
Is the modified application easily maintainable?	

SonarQube's Maintainability Rating considers a set of metrics, including the technical debt ratio, that refers to the cost of developing a line of code. The scale provided by SonarQube goes from A to E, where A is the best value and E is the worst (SonarQube, 2022).

SonarGraph's Maintainability Level considers metrics for coupling and cyclic dependencies to calculate the maintainability level (Zitzewitz Alexander, 2018). It is represented as a percentage and the higher this percentage, the better.

6.2 Experiments

This section details the results of the experiments performed to validate if the energy consumption decreased and to evaluate the effects on the performance and maintainability of the initial and final solutions.

6.2.1 Energy Consumption

After applying some changes that should contribute to a decrease in energy consumption and developing the necessary tests to minimally guarantee that the functionalities were not compromised by the changes, move on to the assessment of energy consumption.

As it was said in section 6.1.1, the metric used to evaluate the energy consumption is the Cumulative Processor Energy, provided by Intel Power Gadget.

Since only one run of each use case could not allow a good understating of a decrease or increase in energy consumption, it was decided to run each use case one hundred times. Although that may not reflect the business context, since some use cases may be used more often than others, to measure energy consumption it is the approach that seems more reasonable. This way, since JMeter allows the creation of thread groups that execute actions as many times as the user wants, it was used to help construct the plan.

After setting up the plan in JMeter, that plan was executed twenty times, and the CPU's energy consumption was tracked in each one of the runs. The computer used to perform the measurements was an Asus with Microsoft Windows 10 Home operational system, with an Intel Core i7-7500U CPU @ 2.70GHz and 8GB of RAM. All the measures were taken with the computer in the same conditions.

Table 15 Results of energy consumption for the initial and final versions

	Initial Application	**Final Application**
Mean	76.18 J	43.58 J
Min Value	64.52 J	37.65 J
Max Value	85.34 J	54.86 J

The results obtained for each one of the twenty runs can be seen in Appendix C. Table 15 presents the results for the initial and the modified application. The values presented in the table represent the mean value, considering all runs as well as the minimum and maximum values measured. As it is possible to verify, there is a significant decrease in energy consumption, validating what the bibliography defends. There was a decrease of approximately 43%, considering the mean values.

Looking at these results we can conclude that the application of techniques to reduce the energy consumption of software resulted in a decrease.

6.2.2 Performance

The performance was measured for the initial and the modified versions of the application, as described in section 6.1.2. The metrics used to evaluate are the Throughput and Total Elapsed Time, provided by JMeter. The test plan used was the same as described previously (section 6.2.1). The test plan runs all the use cases one hundred times.

Table 16 Results of performance for the initial and final versions

	Initial Application		**Final Application**	
	Throughput	Total Elapsed Time	Throughput	Total Elapsed Time
Mean	490.25 /s	3.48 s	596.65 /s	2.86 s
Min Value	377.69 /s	3.26 s	528.44 /s	2.64 s
Max Value	521.79 /s	4.50 s	644.67 /s	3.22 s

The results obtained for each one of the twenty runs can be seen in Appendix D. Table 16 presents the results for the initial and the modified application, respectively. The values presented in the table represent the mean value, considering all runs as well as the minimum and maximum values measured.

As it is possible to verify, there is an increase of around 22% in the throughput, which is positive since it represents the number of requests the application can serve per second. Regarding the total elapsed time, there was a decrease of approximately 18%, considering the mean values.

Looking at these results we can conclude that the application of techniques to reduce the energy consumption of software resulted in an improvement in the performance of the application.

6.2.3 Maintainability

The maintainability was measured for both versions of the application, as described in section 6.1.3. The metrics used to evaluate are the Maintainability Rating, provided by SonarQube, and, Maintainability Level, provided by SonarGraph.

Table 17 Results of maintainability for the initial and final versions

	Maintainability Rating	**Maintainability Level**
Initial Application	A	88.09%
Final Application	A	88.13%

Table 17 presents the results of the initial and the modified application, respectively.

To obtain the SonarQube's Maintainability Rating, the SonarQube was installed locally, and two projects were created, one for each solution. The evaluation is done automatically after deploying to the tool using the command: *mvn clean verify sonar:sonar -Dsonar.projectKey=KEY -Dsonar.host.url=http://localhost:9000 -Dsonar.login=LOGIN*. To have the most realistic results possible, meaning to know the true impact that the application of the techniques previously described had, none of the code smells suggested by SonarQube were corrected.

As referred to in section 6.1.3, SonarQube's Maintainability Rating measure considers metrics as the technical debt ratio, which refers to the cost of developing a line of code, and the number and cost of correcting the code smells. Both solutions present a value of A, so it is needed to look at the sub-metrics used to calculate the Maintainability Rating, to verify if there was any improvement or regression. Table 18 describes those values and it's possible to conclude that there was a slight improvement (below 15%).

Table 18 Values of the maintainability metrics for the initial and final versions

	Initial Application	**Final Application**
Number of Code Smells	122	107
Debt	2d 2h	2d
Debt Ratio	1.7%	1.6%

Moving to the SonarGraph's Maintainability Level, this tool considers metrics for coupling and cyclic dependencies to calculate the maintainability level, as described in section 6.1.3. As stated in Table 17 there is also a slight improvement. This was already expected since the techniques applied do not interfere a lot with the dependencies of the components. However, some of them had an impact since, for example, some processing was moved to other classes in the Feature Envy Technique. Also, some unnecessary calls to methods were removed when applying the Excessive Method Calls Technique, which can also cause a reduction of the dependencies.

6.3 Summary

The first analysis performed was on the energy consumption values. The result of this analysis would corroborate, or not, the studied literature (section 2.4). To measure the energy consumption Intel Power Gadget and JMeter were used (section 6.2.1). As it was possible to verify, a decrease of 43% occurred. This is considered a significant decrease, corroborating the literature.

After validating the decrease in energy consumption, it is needed to analyze the performance and maintainability quality attributes, to answer the research questions (section 1.3).

Regarding performance, the metrics used to evaluate were the Throughput and the Total Elapsed Time, provided by JMeter (section 6.2.2). Observing the Throughput, it was possible to verify an improvement of, approximately, 22%. Regarding the Total Elapsed Time, a decrease of 18% was observed. This is considered a significant improvement in the performance of the application.

Regarding maintainability, the metrics used to evaluate were the Maintainability Rating, provided by SonarQube, and, Maintainability Level, provided by SonarGraph. The first one did not allow any conclusion, so it was needed to consider the sub-metrics used to reach the Maintainability Rating. Even so, the improvements verified were below 15%. Regarding the Maintainability Level, the improvement verified is also not relevant.

This study confirmed that the application of techniques reduced energy consumption and had a significant impact on the performance of the application. Regarding maintainability, the improvements verified are negligible.

7 Conclusions

This section summarizes what was done, difficulties found, threats to validity, future work, and contributions.

7.1 Achievements

This document presents all the steps taken to evaluate the impact that applying techniques to reduce energy consumption had on the maintainability and performance of the application.

To perform this study some objectives were defined (section 1.3):

- **Study how to measure or estimate the energy consumption of software:** research on the existing tools was successfully performed (section 2.2), and some limitations were found. The hardware-based tools were either not available or did not guarantee accuracy. Only the software-based approaches were left and, considering those, only one was possible to use because of the operational system of the computer to be used in this project (section 4.1).

- **Study the existent approaches to decrease the software's energy consumption:** research on the existing techniques to reduce energy consumption was successfully performed (section 2.4). This analysis was done considering a Java backend application, so the techniques documented are only those that could be applied to this scenario.

- **Perform some experiments in an application:** when choosing the software to perform the experiments, due to the available time to develop this project, it was prioritized the knowledge already had of the application (section 4.2). After that, all the code was analyzed to find places where the techniques previously identified could be applied. Not all the techniques were used since some of them did not apply to the selected project (section 5.2). After identifying all the places that needed to be corrected, the corrections were performed, and unit tests were developed to guarantee that the corrections did not impact the behavior of the application.

- **Evaluation of the results:** to plan the evaluation of the solutions (chapter 0), the GQM method was used, which helped to define the metric to be used to evaluate. These metrics were measured using the Intel Power Gadget, JMeter, SonarQube, and SonarGraph tools, and after some research, it was identified that they could assess the elements to analyze, namely, energy consumption, performance, and maintainability. To measure energy consumption and performance, a test plan was created in JMeter. To measure the maintainability, both with SonarQube and SonarGraph, it was only needed to install the tools and import the projects.

In conclusion, with the results obtained from the energy consumption analysis, it was possible to conclude that the application of the techniques had a significant impact on energy consumption. Regarding the study of the impact on performance, it was also possible to verify significant improvements in both metrics: Throughput and Total Elapsed Time. Finally, regarding maintainability, the two solutions presented good results, in both metrics: Maintainability Level and Maintainability Rating. The final version of the application presented slightly better results, but it was considered negligible.

7.2 Threats to Validity

This work is based on the measurement of the CPU energy consumption when running an application. Despite being very careful, some aspects can be considered threats to the validity.

When selecting the tool to use to measure energy consumption, one of the aspects that were considered was accuracy. Intel Power Gadget seems to be a reliable tool and as explained before, some studies already tested its accuracy, but it's important to mention that the results presented here depend on that accuracy.

Also, as explained when listing all the tools available, to use Intel Power Gadget, there is the need to click on a button to start the count and to click again to stop the count. As expected, although there was a lot of caution, some human error could have happened, causing the measurements to possibly have a small margin of error.

Additionally, the hardware specifications can also interfere with the energy consumed. As stated before, the computer used to perform the measurements was an Asus with Microsoft Windows 10 Home operational system, with an Intel Core i7-7500U CPU @ 2.70GHz and 8GB of RAM. The same experiments can have different values when using different hardware.

Plus, when measurements were being taken, care was taken to ensure that the computer was always in the same condition. This way, when measuring the computer was always with the battery full and plugged into the socket. Also, only the necessary programs were running.

Finally, the results presented in this study are dependent on the application used. It is not possible to guarantee that the results would be the same when using another application.

7.3 Future Work

The energy consumption of software is not a simple topic since it involves understanding how different software components and processes consume energy and the factors that influence their energy use. Also, the hardware where the software runs can also affect the consumption.

In the future, it would be interesting to analyze which of the applied techniques affected each one of the quality attributes. Since this study analyzed only the initial and the final versions of the application, it's not possible to conclude the effect of each one of the techniques individually.

Additionally, it would be essential to analyze the effects that this transformation had on other quality attributes. Since both the initial and the modified versions of the application is available on GitHub, they can be used in future works to analyze other elements.

Also, it would be interesting to replicate this experiment with a corporate application (a company that wanted to make its software more sustainable), to evaluate the time the transformation consumed, and the challenges found. Also, in that case, depending on the application and what the company considers critical to analyze, it may be important to analyze more, or other, quality attributes.

Finally, this dissertation helped in the clarification of some questions about how energy consumption can be reduced and the impacts it can have in terms of performance and maintainability. Even so, there are still some points that can be explored.

7.4 Contributions

This document was developed in the context of a master's thesis at ISEP to study the impact of applying techniques to reduce energy consumption in the maintainability and performance of the software.

A description of some techniques that can be used to decrease the energy consumption of software was presented, being more focused on Java applications, since it was the software used. The focus of the study is on evaluating the impact that those techniques can have on some quality attributes. In this study, the attributes chosen were performance and maintainability. The results have shown that there was an improvement in both quality attributes.

The source code of the initial and final versions of the application is available on GitHub, fully open source, for testing, improvements, or to perform another analysis.

References

Acar, H., Alptekin, G. I., Gelas, J.-P., & Ghodous, P. (2016). The Impact of Source Code in Software on Power Consumption. In *International Journal of Electronic Business Management* (Vol. 14).

Alders, R. (2016). *Energy Efficient Design Patterns*.

Amazon. (2023). *Amazon.de : power meter for socket*. https://www.amazon.de/s?k=power+meter+for+socket&s=relevanceblender&ds=v1%3AqTKWFtwfsAN7OvE6v7xkGQmZo6uHcQnedtjF%2FOeoqZ0&crid=2VIIMUD7U3209&qid=1674931803&sprefix=power+meter+for+socke%2Caps%2C444&ref=sr_st_relevanceblender

Basili, V. R., Caldiera, G., & Rombach, H. D. (1994). The Goal Question Metric Approach. In *Encyclopedia of software engineering* (pp. 528–532).

Capra, E., Formenti, G., Francalanci, C., & Gallazzi, S. (2010). *The Impact of MIS Software on IT Energy Consumption*.

Couto, M., Maia, D., Saraiva, J., & Pereira, R. (2020). *On Energy Debt: Managing Consumption on Evolving Software*. https://doi.org/10.1145/3387906

Cruz, L., Abreu, R., Grundy, J., Li, L., & Xia, X. (2019). Do Energy-Oriented Changes Hinder Maintainability? *Proceedings - 2019 IEEE International Conference on Software Maintenance and Evolution, ICSME 2019*, 29–40. https://doi.org/10.1109/ICSME.2019.00013

Ghaleb, T. A. (2019, May 15). Software energy measurement at different levels of granularity. *2019 International Conference on Computer and Information Sciences, ICCIS 2019*. https://doi.org/10.1109/ICCISci.2019.8716456

Green Software Foundation. (2022). *Green Software Foundation*. We Are Building a Trusted Ecosystem of People, Standards, Tooling and Best Practices for GREEN SOFTWARE. https://greensoftware.foundation/

Hirst, J. M., Miller, J. R., Kaplan, B. A., & Reed, D. D. (2013). Watts Up? Pro AC Power Meter for Automated Energy Recording A Product Review. In *Behavior Analysis in Practice* (Vol. 6, Issue 1). http://www.wattsupme-

iea. (2022). *Net Zero Emissions by 2050 Scenario (NZE) – Global Energy and Climate Model – Analysis - IEA*. https://www.iea.org/reports/global-energy-and-climate-model/net-zero-emissions-by-2050-scenario-nze

Intel. (2022, December 10). *Intel Power Gadget*. Intel Power Gadget. https://www.intel.com/content/www/us/en/developer/articles/tool/power-gadget.html

ISO 25000. (2022). *ISO 25000 STANDARDS*. The ISO/IEC 25000 Series of Standards. https://iso25000.com/index.php/en/iso-25000-standards

ISO 25010. (2022). *ISO 25010*. ISO 25010. https://iso25000.com/index.php/en/iso-25000-standards/iso-25010

Kamiya, G. (2022, September). *Data Centres and Data Transmission Networks – Analysis - IEA*. https://www.iea.org/reports/data-centres-and-data-transmission-networks

Kasper Groes Albin Ludvigsen. (2023, March 5). *Medium*. ChatGPT's Electricity Consumption, Pt. II. https://kaspergroesludvigsen.medium.com/chatgpts-electricity-consumption-pt-ii-225e7e43f22b

Khan, K. N., Hirki, M., Niemi, T., Nurminen, J. K., & Ou, Z. (2018). RAPL in action: Experiences in using RAPL for power measurements. *ACM Transactions on Modeling and Performance Evaluation of Computing Systems*, *3*(2). https://doi.org/10.1145/3177754

King, C. (2015, August 16). *Ubuntu Manpage: powerstat - a tool to measure power consumption*. Powerstat. https://manpages.ubuntu.com/manpages/xenial/man8/powerstat.8.html

Koen, P. A., Ajamian, G. M., Boyce, S., Clamen, A., Fisher, E., Fountoulakis, S., Johnson, A., Puri, P., & Seibert, R. (2002). *FuzzyFrontEnd: Effective Methods, Tools, and Techniques LITERATURE REVIEW AND RATIONALE FOR DEVELOPING THE NCD MODEL*.

Maia, D., Couto, M., Saraiva, J., & Pereira, R. (2020). *E-Debitum: Managing Software Energy Debt*. *17*. https://doi.org/10.1145/1122445

Microsoft. (2010, February 23). *Joulemeter: Computational Energy Measurement and Optimization*. Joulemeter: Computational Energy Measurement and Optimization. https://www.microsoft.com/en-us/research/project/joulemeter-computational-energy-measurement-and-optimization/

Monsoon Solutions, I. (2022, December 19). *HIGH VOLTAGE POWER MONITOR*. HIGH VOLTAGE POWER MONITOR. https://www.msoon.com/high-voltage-power-monitor

Mynatt, E. D., Schoner, D., Fitzpatrick, G., SIGCHI (Group : U.S.), Association for Computing Machinery, & ACM Digital Library. (2010). *CHI '10 Extended abstracts on Human Factors in Computing Systems*.

Naumann, S., Dick, M., Kern, E., & Johann, T. (2011). The GREENSOFT Model: A reference model for green and sustainable software and its engineering. *Sustainable Computing: Informatics and Systems*, *1*(4), 294–304. https://doi.org/10.1016/j.suscom.2011.06.004

Nicola, S. (2022). *ANÁLISE DE VALOR INESC-TEC*. https://moodle.isep.ipp.pt/pluginfile.php/240297/mod_resource/content/1/An%C3%A1lise_Valor_Aula_4_21NOV_2018_1hora_AHP.pdf

Noureddine, A. (2022, June 7). *PowerJoular and JoularJX: Multi-Platform Software Power Monitoring Tools*. PowerJoular and JoularJX: Multi-Platform Software Power Monitoring Tools. https://www.noureddine.org/research/joular/powerjoular

Noureddine, A., Islam, S., & Bashroush, R. (2016). *Jolinar: Analysing the Energy Footprint of Software Applications (Demo)*. https://www.noureddine.org/research/jolinar

Noureddine, A., Rouvoy, R., & Seinturier, L. (2015). Monitoring energy hotspots in software: Energy profiling of software code. *Automated Software Engineering*, *22*(3), 291–332. https://doi.org/10.1007/s10515-014-0171-1

Pereira, R., Carção, T., Couto, M., Cunha, J., Fernandes, J. P., & Saraiva, J. (2020). SPELLing out energy leaks: Aiding developers locate energy inefficient code. *Journal of Systems and Software*, *161*. https://doi.org/10.1016/j.jss.2019.110463

Pereira, R., Couto, M., Saraiva, J., Cunha, J., & Fernandes, J. P. (2016). The influence of the Java collection framework on overall energy consumption. *Proceedings - International Conference on Software Engineering*, 15–21. https://doi.org/10.1145/2896967.2896968

Principles.Green. (2022). *Principles of Green Software Engineering*. Principles of Green Software Engineering. https://principles.green/#heading-everyone-has-a-part-to-play-in-the-climate-solution.

Rosencrance, L. (2022, August). *What is green software?* Green Software Definition. https://www.techtarget.com/searchsoftwarequality/definition/green-software

Saaty, T. L. (1990). How to make a decision: The analytic hierarchy process. *European Journal of Operational Research*, *48*(1), 9–26. https://doi.org/10.1016/0377-2217(90)90057-I

Sandquist, J. (2021, May 25). *Accenture, GitHub, Microsoft and ThoughtWorks launch the Green Software Foundation with the Linux Foundation to put sustainability at the core of software engineering*. Accenture, GitHub, Microsoft and ThoughtWorks Launch the Green Software Foundation with the Linux Foundation to Put Sustainability at the Core of Software Engineering. https://blogs.microsoft.com/blog/2021/05/25/accenture-github-microsoft-and-thoughtworks-launch-the-green-software-foundation-with-the-linux-foundation-to-put-sustainability-at-the-core-of-software-engineering/

Şanlıalp, İ., Öztürk, M. M., & Yiğit, T. (2022). Energy Efficiency Analysis of Code Refactoring Techniques for Green and Sustainable Software in Portable Devices. *Electronics (Switzerland)*, *11*(3). https://doi.org/10.3390/electronics11030442

SonarQube. (2022). *Metric Definition*. https://docs.sonarqube.org/latest/user-guide/metric-definitions/

Spirals research group. (2022, December 7). *PowerAPI*. PowerAPI. http://powerapi.org/

United Nations. (2022a). *Goal 7 | Department of Economic and Social Affairs*. Ensure Access to Affordable, Reliable, Sustainable and Modern Energy for All. https://sdgs.un.org/goals/goal7

United Nations. (2022b). *The Paris Agreement | UNFCCC*. https://unfccc.int/process-and-meetings/the-paris-agreement/the-paris-agreement

Vedantu. (2022, December 28). *Energy Consumption Formula - Meaning, Calculation and FAQs*. https://www.vedantu.com/formula/energy-consumption-formula

Verdecchia, R., Aparicio Saez, R., Procaccianti, G., & Lago, P. (2018). *Empirical Evaluation of the Energy Impact of Refactoring Code Smells* (Vol. 52).

World Health Organization. (2022). *Sustainable Development Goals*. https://www.who.int/europe/about-us/our-work/sustainable-development-goals

Zettelmeyer, J., Tagliapietra, S., Zachmann, G., & Heussaff, C. (2022, December). *Beating the European Energy Crisis*. https://www.imf.org/en/Publications/fandd/issues/2022/12/beating-the-european-energy-crisis-Zettelmeyer

Zitzewitz Alexander. (2018, October 10). *A Promising New Metric To Track Maintainability – hello2morrow – Empowering Software Craftsmanship*. https://blog.hello2morrow.com/2018/12/a-promising-new-metric-to-track-maintainability/

Appendix A: Authorization to use the Selected Project

Tese no Âmbito do Mestrado em Engenharia Informática

AR **Ana Ribeiro (1170972) <1170972@isep.ipp.pt>**
11/02/2023 14:38

Para: André Alves; Jose Ferreira (1171066) Cc: Isabel Azevedo

Boa tarde,

Estou atualmente a realizar a minha tese no âmbito do Mestrado em Engenharia Informática, ramo Engenharia de Software.

A tese a ser desenvolvida tem como objetivo a análise do impacto que a adoção de técnicas de redução do consumo energético de software tem nos atributos de qualidade manutenibilidade e performance. Desta forma, necessito de um projeto base para medir os atributos de qualidade referidos, aplicar algumas transformações e voltar a efetuar uma medição.

Uma vez que durante o mestrado já foram realizados alguns trabalhos, e pela vantagem de já ter conhecimento sobre o software, gostaria de utilizar o trabalho que foi desenvolvido no âmbito da unidade curricular de Arquitetura de Software, denominado Gorgeous Sandwich.

Desta forma, envio este email com o intuito de obter a permissão da utilização do projeto desenvolvido por mim, André Alves e José Ferreira e avaliado pela professora Isabel Azevedo, também orientadora da tese.
O projeto terá de ser publicado no GitHub, de forma a estar visível para todas as pessoas envolvidas na escrita e avaliação da tese.

Fico a aguardar uma resposta,
Cumprimentos,
Ana Catarina Ribeiro

Figure 14 Request to use the project

Re: Tese no Âmbito do Mestrado em Engenharia Informática

IA **Isabel Azevedo <ifp@isep.ipp.pt>**
11/02/2023 14:41

Para: Ana Ribeiro (1170972); André Alves; Jose Ferreira (1171066)

Boa tarde, Catarina.

Nenhum problema da minha parte, desde que o André Alves e o José Ferreira autorizem.

Cumprimentos,
Isabel Azevedo

Figure 15 Authorization from Teacher Isabel Azevedo

RE: Tese no Âmbito do Mestrado em Engenharia Informática

JF **Jose Ferreira (1171066) <1171066@isep.ipp.pt>**
11/02/2023 15:17

Para: Isabel Azevedo; Ana Ribeiro (1170972); André Alves

Boa tarde,

Da minha parte também não existe qualquer problema, eu autorizo a utilização do projeto Gorgeous Sandwich, desenvolvido no âmbito da unidade curricular de Arquitetura de Software.

Espero que todos se encontrem bem! Continuação de um bom trabalho e boa sorte para a realização da tese Catarina!

Cumprimentos,
José Ferreira

Figure 16 Authorization from Student José Ferreira

Re: Tese no Âmbito do Mestrado em Engenharia Informática

AA **André Alves <andrefralves1999@gmail.com>**
11/02/2023 16:01

Para: Jose Ferreira (1171066) Cc: Isabel Azevedo; Ana Ribeiro (1170972)

Boa tarde,

Do meu lado não existe problema também, eu autorizo a utilização do projeto.

Cumprimentos,
André Alves

Figure 17 Authorization from Student André Alves

Appendix B: Faults detected

Table 19 Java Collection Framework

Interface	Class	Method/Variable	Implementation	Observations
Set	OrderConverter	convertOrderItemsListToDTO()	HashSet	The only method used is the add. HashSet is the implementation that consumes more energy in the add execution. The implementation that consumes less energy is the TreeSet. The HashSet should be replaced with a TreeSet.
		convertOrderItemsListFromDTO()	HashSet	
Set	OrderMapperJPA	convertToDomain()	HashSet	
Map	Type	nameIndex	HashMap	The methods used are get and put. Hashtable is the implementation that consumes less energy in both methods. The HashMap should be replaced with a Hashtable.
Map	OrderMapperJPA	convertToPersistence()	HashMap	

Table 20 Excessive Method Calls

Class	Method	Observations
OrderRepositoryWrapperJPA	update()	Uses method existsById() but it is already checked in the controller.
SandwichRepositoryWrapperJPA	update()	
Sandwich	changeStock()	Calls the obtainUnits() method multiple times. The return of the method should be stored in a variable and the variable should be used.
OrderController	updateOrder()	These methods use dto.deliveryTime.start and dto.deliveryTime.end multiple times. The dto.deliveryTime return should be assigned to a variable and the variable should be used.
OrderConverter	convertToDTO()	
	convertFromDTO()	
OrderMapperJPA	convertToPersistence()	

Table 21 Encapsulate Field

Class	**Variable**	**Observations**
Grade	MIN_VALUE	Public fields that are written in GorgeousSandwichApplication. Need to implement the setter method.
	MAX_VALUE	

Table 22 Feature Envy

Class	**Method**	**Observations**
OrderController	checkIfSandwichExists()	Multiple calls to the database. Implement only one query that accepts an array of Ids.
DeliveryTime	calculateIntervals()	Every time an order is created, or updated, or a get of the delivery times is performed, this method runs. Since the variables to run this method are read in the main function, the calculation of the delivery times should be done there, to be done just once.
OrderRepositoryWrapperJPA	update()	The validation of the delivery time should be done before.

Table 23 Member Ignoring Method

Class	**Method**
DeliveryTime	changeDeliveryTimes()
OrderController	getDeliveryTimes()
CommentConverter	convertToDTO()
	convertFromDTO()
	convertCommentListToDTO()
OrderConverter	convertToDTO()
	convertFromDTO()
	convertOrderItemToDTO()
	convertOrderItemFromDTO()
	convertOrderItemsListToDTO()
	convertOrderItemsListFromDTO()
	convertListToDTO()
ReviewConverter	convertToDTO()
	convertFromDTO()
	convertReviewListToDTO()
SandwichConverter	convertToDTO()
	convertFromDTO()
	convertListToDTO()
CommentMapperJPA	convertToDomain()
	convertToPersistence()
	convertListToDomain()
	convertListToPersistence()
OrderMapperJPA	convertToDomain()
	convertToPersistence()
	convertListToDomain()
ReviewMapperJPA	convertToDomain()
	convertToPersistence()
	convertListToDomain()
SandwichMapperJPA	convertToDomain()
	convertToPersistence()
	convertListToDomain()

Table 24 Inline Temp and Introduce Explaining Variable

Class	**Method**
CommentController	listBySandwich()
	listByEmail()
	createComment()
OrderController	listAll()
	getById()
	getByEmail()
	createOrder()
	updateOrder()

ReviewController	listBySandwich()
	listByEmail()
	createReview()
SandwichController	listAll()
	getById()
	createSandwich()
	addUnitsSandwich()
Comment	equals()
CommentId	equals()
DeliveryDate	equals()
DeliveryTime	equals()
Order	changeDeliveryTime()
	equals()
OrderDate	equals()
OrderId	equals()
OrderItem	equals()
OrderStatus	equals()
Quantity	equals()
Grade	equals()
Review	equals()
ReviewId	equals()
Designation	equals()
Sandwich	equals()
SandwichId	equals()
Stock	equals()
Description	equals()
UserEmail	equals()
CommentRepositoryWrapperJPA	save()
OrderRepositoryWrapperJPA	save()
	findAll()
	update()
ReviewRepositoryWrapperJPA	save()
SandwichRepositoryWrapperJPA	save()
	findAll()
	update()

Appendix C: Energy Consumption Results

Table 25 Results of energy consumption of the initial application for each one of the runs

Run	Energy Consumption
1	77,25 J
2	72,77 J
3	76,59 J
4	76,80 J
5	70,28 J
6	72,05 J
7	75,98 J
8	78,77 J
9	75,30 J
10	72,32 J
11	81,75 J
12	73,11 J
13	85,34 J
14	77,46 J
15	74,72 J
16	80,21 J
17	83,99 J
18	81,15 J
19	73,28 J
20	64,52 J

Table 26 Results of energy consumption of the modified application for each one of the runs

Run	Energy Consumption
1	53,25 J
2	46,52 J
3	54,86 J
4	48,13 J
5	47,41 J
6	43,93 J
7	37,65 J
8	48,52 J
9	40,36 J
10	40,70 J
11	40,24 J
12	42,71 J
13	40,51 J
14	40,77 J
15	40,58 J
16	40,23 J
17	41,03 J
18	41,13 J
19	40,91 J
20	42,12 J

Appendix D: Performance Results

Table 27 Results of performance of the initial application for each one of the runs

Run	Throughput	Total Elapsed Time
1	483,78 /s	3,51 s
2	493,04 /s	3,45 s
3	497,66 /s	3,42 s
4	468,06 /s	3,63 s
5	491,33 /s	3,46 s
6	377,69 /s	4,50 s
7	485,71 /s	3,50 s
8	508,83 /s	3,34 s
9	509,44 /s	3,34 s
10	480,77 /s	3,54 s
11	516,25 /s	3,29 s
12	505,20 /s	3,37 s
13	512,67 /s	3,32 s
14	496,64 /s	3,42 s
15	495,19 /s	3,43 s
16	496,50 /s	3,42 s
17	521,79 /s	3,26 s
18	495,34 /s	3,43 s
19	496,21 /s	3,43 s
20	472,88 /s	3,60 s

Table 28 Results of performance of the modified application for each one of the runs

Nº	**Throughput**	**Total Elapsed Time**
1	545,75 /s	3,12 s
2	528,44 /s	3,22 s
3	530,92 /s	3,20 s
4	562,73 /s	3,02 s
5	587,63 /s	2,89 s
6	606,93 /s	2,80 s
7	550,70 /s	3,09 s
8	600,92 /s	2,83 s
9	619,76 /s	2,74 s
10	609,32 /s	2,79 s
11	625,46 /s	2,72 s
12	644,67 /s	2,64 s
13	603,69 /s	2,82 s
14	613,50 /s	2,77 s
15	624,08 /s	2,72 s
16	605,84 /s	2,81 s
17	611,95 /s	2,78 s
18	608,88 /s	2,79 s
19	630,10 /s	2,70 s
20	621,80 /s	2,73 s