Green Code

Second extended edition

Janne Kalliola, Exove

Foreword by Professor Jari Porras, LUT University

EXOVE

Green Code

Second extended edition

Version 2023-08-31

Do not print this book.
The carbon footprint of the paper needed is about 600 gCO_2eq.[1]

ISBN: 978-952-65306-2-8

[1] www.sciencedirect.com/science/article/abs/pii/S0959652611004409

Janne Kalliola

Janne Kalliola is a founder of Exove, where he currently serves as Chief Growth Officer.

He is also the chairman of the board of Code from Finland association, and has launched the development of a carbon neutrality label.

Janne has been coding since 1983, and has published several commercial software products both internationally and in Finland.

He is a frequent speaker at open source and green ICT events, and is passionate about eco-efficiency and green coding.

kallio.la

linkedin.com/in/jannekalliola

twitter.com/plastic

Exove

Exove is a software company that combines analytical and technological know-how with human insight. The company focuses on building digital solutions that fight against digital frustration – defrustrating the digital.

Some of our most important clients include Neste, Sanoma, Loiste, Rukakeskus, the University of Oulu, the University of Eastern Finland, LUT University, JAMK University of Applied Science, and the cities of Tampere and Jyväskylä.

exove.com

linkedin.com/company/exove

twitter.com/exove

Exove is part of the Finnish PunaMusta Media group, which employs around 810 professionals from various fields. The group's revenue in 2022 was 133.5 million euros. PunaMusta Media is committed to setting Science Based Targets (SBT) for driving sustainability and reducing climate impact.

Exove Sustainability Compass

We have created our own sustainability program – The Sustainability Compass – which leads our actions both internally and externally.

We want to lead the way in our field and improve the sustainability of our projects by creating and implementing guidelines for social and environmental sustainability.

These four directions cover our impact over people and planet, with our clients and internally.

1. Sustainable web design and development
2. Social sustainability
3. Environmental sustainability
4. Responsible company governance

www.exove.com/sustainability/

Contents

1 Foreword

“We must change almost everything in our current societies.
The bigger your carbon footprint - the bigger your moral duty.
The bigger your platform - the bigger your responsibility.
Adults keep saying: 'We owe it to the young people to give them hope.'
But I don't want your hope.
I don't want you to be hopeful.
I want you to panic.
I want you to feel the fear I feel every day.
And then I want you to act.
I want you to act as you would in a crisis.
I want you to act as if our house is on fire.
Because it is.”

— Greta Thunberg, No One Is Too Small to Make a Difference

Climate change has been a major topic in political debate in recent years. The Paris climate conference (COP21) in 2015, which set a target of limiting global warming to below 1.5°C compared to pre-industrial levels, sparked a conversation about what different industries can do to help achieve this target. In the same year, the Global e-Sustainability Initiative's #SMARTer2030 report outlined a path forward for the ICT industry in addressing these challenges. This book focusing on green code is a natural continuation of this conversation related to possibilities of

different industries to address climate change. The subject is important for many reasons, as you will find out when you read the book.

The author of the book, Janne Kalliola, approaches the subject from his own experience as a young programmer in the 1980s. At the time, both devices and programming environments were still in their early stages and had limited memory and processing resources. This influenced the way people wrote code. Although the industry has advanced significantly since then, there are still many constraints that need to be taken into account in writing efficient code, particularly in limited resources in embedded systems and execution times in high-performance computing. So why does Janne Kalliola emphasise the importance of green code and what exactly is green coding?

In the book, green code refers to the energy consumption of software and resulting carbon footprint and handprint. The perspective is challenging for many reasons.

The energy required by software depends on many factors, as discussed in the book. For example, the choice of algorithm, the programming language and external libraries used, and the chosen development and runtime environments all affect the efficiency of the executed code. These decisions are interdependent, which complicates the task of making optimal choices.

Measuring the energy consumption of software without the runtime environment affecting the measurements is difficult. Seemingly identical devices, such as smartphones, can have significant differences in energy consumption depending on their implementation.

The energy efficiency of software is not a critical factor – with the possible exception of embedded systems – such as, for example, the size of the software was in the early days of computing. If a software used too much memory, it was impossible or at least very difficult to run it on the early-day platforms. When the goal of software efficiency is to affect emissions and ultimately climate change, the chain of effects relating to

both functionalities and the time frame becomes difficult to understand and observe.

The book presents reasons for the inefficiency of current code and solutions as well as recommendations for making code more efficient. In my opinion, the book's main message is that as the constraints of the programming environment have diminished, the quality of software has deteriorated over time, leading to unnecessary energy consumption. To quote the book's author, "In Finland, the focus is on writing code efficiently instead of writing efficient code." The energy efficiency of software can be improved as soon as energy efficiency becomes an essential requirement for software.

This book by Janne Kalliola serves as a good discussion opener for this topic, but it is important for the development of the field that the discussion and development work continue after this.

In Lappeenranta on November 23rd, 2022
Professor Jari Porras
LUT University

2 Introduction

When I began my programming career with the Commodore VIC-20 I received for Christmas, efficiency was a crucial factor from the start. The computer had an eight-bit processor with a clock speed of one megahertz and only five kilobytes of memory. If the code wasn't efficient, it wasn't very practical either. Five kilobytes didn't allow for very complex applications, and the Basic interpreter I used had only 3.5 kilobytes of memory available.

For comparison, my phone has a 64-bit octa-core processor with a clock speed of 2.9 GHz and eight gigabytes of memory. This means it has 23,200 times more processing power (not considering the extra 56 bits the processor processes with each cycle or the pipeline of modern processors) and 1.6 million times more memory. Of course, I can do more with my phone than I could with the early, inexpensive computer.

Even when compared to my first Amiga 500, which had a graphical operating system, all the necessary utility software, and many great games, the ratios don't change much. The Amiga had a 7.16 Mhz 16-bit processor and 512 kilobytes of main memory – my phone has 3,318 times more CPU power and 16,384 times more memory.

Many people think that time must have gilded my memory. It is partly true, as the ancient software was sometimes slow and had fewer features, and the systems crashed more often. However, my beloved Amiga is still in storage and a couple of years ago I showed my children what information technology was like when I was their age. The session ended

when the children left for their mobile phones, and I continued to play Populous for a few hours. The gaming experience was similar to the mobile games I've been playing, although Populous didn't force me to watch ads or ask for money.

None of the current applications would be able to run on the old hardware. They wouldn't fit in the memory, and they would be so slow that the user would lose patience quickly. Yet, it was possible to develop applications that achieved more or less the same things as current ones. Where does this need for power really come from?

Of course, new software has features that need the power of current devices. For example, Photoshop has many automatic correction tools that work exceptionally well and save hours of manual work. Additionally, file sizes have increased, requiring more memory and power to process them.

A lot of current software is bloated only because there is no need to focus on the efficiency of applications. Hardware is cheap, and the network has plenty of transmission capacity, so there is no incentive to optimise. As this situation has persisted for over a decade, much of the modern code has never been optimised with care. It has always been fast enough. Optimisation is also an expensive undertaking and doesn't necessarily pay for itself in terms of investment. The savings may be very small.

The situation is now changing. Climate change and the energy crisis in Europe, caused by Russia's aggressive war, are forcing us to reexamine all energy consumption. Software must also change along with the rest of the world. This will be a long, complicated, and certainly painful process, as many of us are writing software in a way that will no longer be acceptable in the future.

Green IT and green code have been widely discussed in Finland over the past couple of years. I have been vocal on the topic, and together with the rest of Code from Finland's board, I have prepared a carbon neutrality label and criteria for the software industry. Several software companies have started to discuss green coding. Finnish Information Society

Development Centre Tieke has initiated the Green ICT project and ecosystem to promote these efforts[2].

But nobody has yet defined what green code or eco-efficient systems are. Everyone looks at the issue from their own perspective, and everyone in the conversation is partly still talking past each other. This is why I wrote this short book and now, almost a year later, I added a section to it about energy-intensive standalone solutions, such as artificial intelligences and crypto contracts.

The purpose of the book is not to unequivocally define green code and create a canon. Instead, I have tried to focus on describing patterns to stimulate thinking and presenting various solutions. By using these, anyone working on the matter can analyse their situation and identify the changes they need to make to become more environmentally friendly.

Finland, as well as other countries, has a long tradition of demo coding from the 90s, and our coders are well-educated. We have practical expertise in writing neat and effective code. Let's use these old ways, incorporate the solutions required by the modern world, and start writing more efficient code. Line by line, application by application. Let's be part of the solution, not part of the problem.

In Espoo on 31st August 2023, while listening to Róisín Murphy's Overpowered.

Janne Kalliola

[2] tieke.fi/en/projects/green-ict-project/

3 Why Should Code Be Green?

The importance of digitalisation for improving the efficiency of operations is undeniable, and it is difficult to imagine today's society without ubiquitous software. The impact of software is significant and is rapidly increasing also in terms of energy consumption.

To combat climate change, all viable avenues must be explored, and the ICT industry must also play a role. For the past twenty years, software has been developed without significant concern for efficiency, as the speed of devices has increased simultaneously. Additionally, more power can easily be obtained from the cloud with a few clicks or even automatically. Scaling with hardware has been cheaper than optimising software.

Processing, presenting, and transferring information consumes energy. At the moment, there is not enough clean energy available globally, and renewable energy sources cause large fluctuations in energy availability and price. Therefore, saving energy makes sense.

Although many data centres operate with renewable or even carbon-neutral energy, this does not change the overall situation. Dirty energy is still produced, it is just used elsewhere. Reducing overall energy consumption and shifting consumption to days when there is a lot of renewable energy available are crucial actions for our planet.

3.1 Handprint and Footprint

When discussing the impact of products, services, or processes on the environment, the term **carbon footprint** is often used. This refers to the amount of emissions produced by the product or service, i.e. how much greenhouse gases are produced during the life cycle of said product or service.

The carbon footprint is measured in carbon dioxide equivalent, which is a measure of all greenhouse gases. It can be used to calculate the impact of various greenhouse gases, such as carbon dioxide or methane, on climate change. Different gases behave differently in the atmosphere, so coefficients have been calculated to allow them to be evaluated on a common scale.

Different forms of energy production have their own carbon footprint, which depends on the way the energy is produced and all the activities required to produce or prepare the production of the energy. Renewable energy is not carbon-neutral in itself, as wind turbines, for example, need to be built, installed, and maintained.

Energy consumption, particularly its growth, is a problem because dirty energy is still being produced, and the production of renewable energy is not sufficient to meet the increasing global energy consumption. Many IT sector companies use renewable or compensated carbon-neutral energy, which reduces the calculated emissions of these companies and encourages the building of more renewable energy capacity. However, reducing consumption is even more important because increased consumption still leads to the use of fossil fuels.

On the other hand, software can also reduce emissions by streamlining or optimising other operations. This is called the **carbon handprint**. As part of the increase in productivity, the IT industry has eliminated unnecessary intermediate steps in processes or for example, minimised the use of paper and printing. This handprint has a significant positive impact on the planet and should not be underestimated. However, its existence does not justify the inefficiency of software and growing energy consumption. The

same handprint can be achieved efficiently or inefficiently, so it makes sense for the world to choose an efficient implementation.

3.2 Trends Lead in the Wrong Direction

There are currently several trends in the IT sector that unfortunately move in the wrong direction when it comes to the climate:

- **Growth of data** – There has been an exponential growth in the amount of data that is produced, processed, and stored. This data generally accumulates, and old data is often not deleted when new data arrives. Instead, it is stored for years or even decades.
- **Measuring software development** – In my own experience, the effectiveness of software development is typically evaluated based on the number of features developed within a specific time frame, regardless of the quality or efficiency of the code. The only time the efficiency of the code becomes a significant factor is when it has a noticeable impact on the user experience.
- **Lust for new devices** – The IT industry is known for its focus on creating more powerful devices, even when the existing technology is enough to meet our needs. This focus on technical specifications can make older devices obsolete quickly, particularly when they are no longer able to receive software security updates or support the increased performance requirements of new applications. As a result, consumers are often forced to upgrade their devices frequently in order to keep up with the latest technology.
- **Transition to mobile networks** – more and more information is transmitted wirelessly and more daily tasks are performed on mobile devices. It is very convenient, but wireless connectivity is significantly less energy-efficient than using a wired connection. On the other hand, mobile phones are more energy-efficient than computers for data processing, Therefore, using a mobile phone instead of a computer can be a good idea if it allows you to avoid using a computer altogether.

- **Advertisement-based funding** – most mobile apps are free to use, and their development is funded by advertising, in-app purchases, or a combination of the two. However, a study[3] by Aalto University in Finland found that the process of conducting automated auctions between advertisers for each ad slot a user sees can be energy intensive, with ad networks consuming an estimated 10% of the total electricity consumed by the entire internet.
- **Growth of artificial intelligence usage** – over the past year, artificial intelligence has evolved from an abstract concept into an everyday tool. New AI-based or machine learning-based solutions emerge daily, often driven by massive amounts of data and utilising significant computational capacity. We still do not know whether the benefits of using artificial intelligence – the enhancement of processes and practices – will outweigh the consumption generated by its use. Or if it will result in yet another vain way of consuming large amounts of energy.

As the amount of software and data continuously grows, faster devices and networks are needed to keep up. This leads to a cycle in which less efficient code or larger amounts of data are used, requiring even faster devices and networks. We should break this cycle.

For example, websites have grown year by year. In 2022, Aalto University conducted a survey[4] of popular Finnish websites. The sample set included both private companies and public administration, totaling around a thousand different sites. The survey found that websites are produced with varying levels of skill. Some of the sites are optimised well and are small in size, while others cause a vast amount of data to be transferred over the network.

[3] www.sciencedirect.com/science/article/pii/S0195925517303505

[4] aaltodoc.aalto.fi/bitstream/handle/123456789/114010/isbn9789526407395.pdf (available in Finnish only)

The HTTP Archive[5] has recorded changes to websites in its annual publication, Web Almanac. Statistics show that the size of websites has tripled in the last ten years[6]. In the past, mobile websites were generally smaller than their desktop counterparts, but today, they are just as large due to the fact that separate mobile versions are no longer implemented. Instead, the appearance of a website changes responsively depending on the capabilities of the user's device.

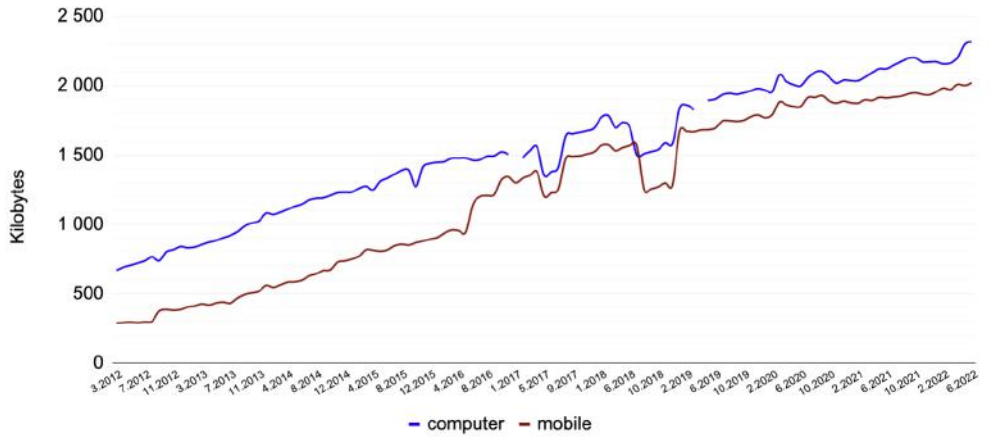


Figure 1. The growth of website size over the years. Source: Web Almanac.

Additionally, consumers' demands for websites and services have also increased. As screen resolutions improve and transmission networks become faster, consumers have come to expect content that looks good on new devices.

According to the aforementioned study by Aalto University, over 40% of the data on web pages was from third-party resources in all measurements. In a sample of a thousand pages, the data was distributed as follows:

5 httparchive.org

6 almanac.httparchive.org/en/2021/page-weight

Type of file	Share	Description
Images	56%	Images used on the site, for example, photos, background images, and icons.
JavaScript	22%	Code to implement features, integrations, and user tracking on the site.
Media files	6,3%	Audio and video files on the site.
Fonts	4,7%	Fonts used on the site. Icons may be also provided as a font.
XHR	4,5%	Files retrieved by JavaScript code on the site. The files may be images, videos, text, or documents for certain applications.
CSS	2,7%	The visual layout building instructions for the pages of the site.
HTML	2,0%	The structure and textual content of pages of the site.
Other files	1,4%	Files that could not be categorised into aforementioned main categories.

The numbers in the table come from the "scrolled computer" category, where pages of the website were viewed on a computer and fully scrolled through. Some resources are only loaded when they become visible (lazy loading). As a result, scrolling can change the relative amounts of data for different file types.

It is worth noting that the text content and structure of a page make up only 1/50 of the page's data. Further, a significant portion of this is used to describe the structure and reference other files. On modern websites, new text content can also be loaded as the user scrolls the site, in which case

these loaded contents are counted under the XHR type, even if they are HTML.

Although images and media are important on a website, most web pages are still primarily text-based and visitors come for the textual content. The ratio of utility is poor.

Over a fifth of a website's data comes from various script files, which make the site more user-friendly and, on the other hand, report the user's activities to analytics and advertising systems.

3.3 Efficient Devices Deceive Developers

Software developers often have very powerful laptops that are only a few years old. On average, business equipment is renewed every three years when the leasing period ends. This may mislead developers into thinking that all devices have this level of power, so they may not realise to optimise their applications adequately. As a result, people using older devices may experience slow performance or may not be able to use the application at all.

Additionally, creating backward-compatible software is more expensive than leaving older devices unsupported. All of these factors can force people to replace their devices, leading to increased emissions from manufacturing and logistics.

However, devices and networks have also become more energy intensive. An increase in processing power does not necessarily require an increase in electricity consumption due to technological progress. On the other hand, devices could be significantly more energy efficient with current technology if software had lower processing requirements.

According to studies[7] by Sitra, electronic waste is the fastest-growing type of waste, with a 7% annual growth rate, and only 17% of it is properly

[7] https://www.sitra.fi/en/articles/five-important-questions-about-the-environmental-impacts-of-increased-digital-use/

treated each year. It should also be noted that recycling itself is a very energy-intensive process, as it involves melting metals, for example. While recycline an obsolete or unnecessary device is better than not recycling it, the best option is to try to extend the life of the device.

To give a sense of the scale of the problem, the same Sitra article calculated that 5,000 Eiffel Towers could have been built with the electronic waste produced in 2019.

3.4 Themes from the European Union

The European Union is one of the world's most advanced communities in matters of responsibility. The Union's recommendations for building more energy-efficient systems have been described in the Official Journal of the European Union[8] published on November 25, 2021. The aim of the document *is to support all organisations in the telecommunications and ICT services sector to focus on relevant environmental aspects, both direct and indirect, and to find information on best environmental management practices, as well as appropriate sector-specific environmental performance indicators to measure their environmental performance, and benchmarks of excellence* – direct quote – and it gives a good picture of the development trends in the EU region.

The interesting topics for this booklet begin with section *3.1.6 Minimising data traffic demand through green software*. The described best practice includes the following actions:

- Develop more energy-efficient software that minimises power consumption of ICT equipment.
- Design demand-adaptive software according to end-user needs, in order to avoid energy overconsumption and to limit the obsolescence of existing devices.

[8] eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32021D2054&from=EN

- Monitor the energy consumption of software.
- Assess environmental impacts of software through life-cycle analysis at development phase and performance measurement at usage phase.
- Refactor existing software to improve its energy efficiency.

Similarly, section *3.2.2 Define and implement a data management and storage policy* is interesting. It describes the best course of action:

- Minimise the amount of stored data.
- Maximise the usage of shared platforms.
- Consolidate existing services and decommission unnecessary hardware.

The document also discusses extensively improving the energy efficiency of data centres and data transmission networks, as well as optimising the energy usage of users' devices.

In addition to the best operating method, each chapter describes a set of *performance indicators* and *benchmarks of excellence*, which can be used to compare your own company with highly advanced organisations. Below are a few selections from the metrics:

- Amount of data transferred in relation to software utilisation (bit/web page view or bit/min of mobile application use)
- Share of newly acquired software for which the energy performance has been used as a selection criterion within procurement (%)
- Share of newly developed software for which the energy performance has been used as a development criterion (%)
- Share of existing software which has been refactored or which has undergone code reviews towards higher energy efficiency (%)

- Share of software developers (staff) trained on energy-efficient software (%)

It is not yet difficult to reach the benchmarks of excellence. The following benchmarks are mentioned for metrics related to software development:

- All staff (software developers) trained on energy-efficient software
- At least one project for minimising data traffic demand through green software was implemented during the year

3.4.1 New Reporting Directive CSRD

The European Union is also updating its own legislation on corporate sustainability reporting.[9] The new directive, the Corporate Sustainability Reporting Directive (CSRD), is currently being developed and will replace the current Non-Financial Reporting Directive (NFRD)[10].

The goal is to fix the shortcomings of the current rules and harmonise reporting to enable comparability. The directive is being developed in response to the need to assess a company's environmental and social impacts and thus measure the sustainability of its business operations. Sustainability of business is reported in a separate sustainability report based on the directive and European sustainability reporting standards. For example, a company's greenhouse gas emissions and their development are disclosed annually.

Under the proposal, large and publicly listed companies will be required to report on sustainability themes in machine-readable format as part of their annual report. The European Commission will subsequently define specific sectors where reporting obligations will also apply to small and

[9] www.consilium.europa.eu/en/press/press-releases/2022/02/24/council-adopts-position-on-the-corporate-sustainability-reporting-directive-csrd/

[10] greenly.earth/en-us/blog/company-guide/what-is-the-non-financial-reporting-directive-nfrd

medium-sized enterprises. The accuracy of the reported information will be audited by the company's auditor.

There will apparently be 13 reporting standards, which will be detailed. You can learn more about the standards and the process on the website of the European Financial Reporting Advisory Group (EFRAG).[11]

3.5 IFRS and Emissions Disclosure

In October 2022, the International Sustainability Standards Board (ISSB) unanimously decided to require companies following IFRS to report their carbon footprint, including emissions under all three scopes of carbon calculation[12].

International Financial Reporting Standards (IFRS) are standards for publishing financial statement information. It has been in use in all EU countries since 2005 and is required in a total of 167 regions worldwide, including practically all developed countries.[13]

In the future, companies reporting in accordance with IFRS will also have to disclose their own emissions as part of their financial statements. Since all three scopes are included, the energy consumption and emissions of IT systems will need to be reported as well.

The requirement is not being enforced immediately, and the exact schedule is currently unknown. It is likely that a transition period of a few years will be granted to allow companies to make the required changes. Nonetheless, especially large companies should start monitoring emissions now and establish control over the energy consumption and emissions of their IT systems piece by piece.

11 www.efrag.org/lab3

12 www.ifrs.org/news-and-events/news/2022/10/issb-unanimously-confirms-scope-3-ghg-emissions-disclosure-requirements-with-strong-application-support-among-key-decisions/

13 en.wikipedia.org/wiki/International_Financial_Reporting_Standards

3.6 Responsibility of Developers

The efficiency of the implementation of an application is primarily the responsibility of software architects and developers. The requirements for the application, which usually arise from business needs, largely determine the efficiency or inefficiency of the application – unfortunately, typically inefficiency.

Only software development professionals have the knowledge and ability to create applications, efficient or inefficient. Unfortunately, some in the industry still believe that software efficiency is not important, perhaps because a software's carbon handprint is large, or because they think other matters are more important.

Energy efficiency is currently not a requirement or selection criteria in the procurement of software, but some organisations are starting to recognise its importance. It is likely that we will see more requirements and outright demands for energy-efficient solutions in the near future. I hope that this book will contribute to the change in procurement criteria to include energy efficiency.

I've had discussions, sometimes fiery, about whether it makes more sense for an application developer to spend time optimising the application or to come to work by bike instead of driving a car. However, these are not mutually exclusive, as even a bike-riding developer can optimise their application, and smart people choose both.

As Spider-Man's uncle Ben Parker has stated in his various incarnations: "with great power comes great responsibility." [14]

Summarised

[14] en.wikipedia.org/wiki/With_great_power_comes_great_responsibility

1. Digitisation is essential for overall efficiency, but the growing energy consumption of software requires attention. The ICT industry must address its own energy efficiency to combat climate change.

2. A critical challenge is the energy consumption of the ICT sector, comprising the energy used by software, devices, and data transmission. Fluctuations and shortages in renewable energy availability necessitate measures to save energy within the ICT sector as well.

3. Trends such as data growth, inefficient software, continuous device upgrades, wireless connections, and ad-based financing of services undermine energy efficiency and increase environmental impacts.

4. Energy-efficient software development is crucial to curb the performance requirements of devices and networks. Developers play a key role in optimising applications to restrain energy consumption.

5. The European Union focuses on sustainability through regulations like CSRD, and similarly, the IFRS standard will mandate carbon reporting. Calculating software carbon footprints will become practically mandatory due to these regulations.

4 What We Know and What We Don't

The energy consumption of software is quite simple, in principle. Data processing requires clock cycles from the processor, and transferring data for processing from memory, network, or storage requires both processor and IO power.

However, the complexity and layering of software, as well as the lack of measurement points, make it difficult to calculate the actual energy consumption. In a single device, there are easily hundreds of processes running and consuming energy, each with its own architecture, algorithms, and data. While the energy used by the device can be measured by the amount of electricity it consumes, the distribution among different components and processes stays unclear and difficult to measure.

In addition, complexity increases when there are multiple devices in the environment, such as an end-user device, a proxy server, a database server, various cache servers, and network devices, each with its own specific software and features. On top of this, there are different abstraction layers, such as virtual machines, containers, and microservice architecture-based implementations.

Measurement data originating from multiple sources is typically also inconsistent, rendering the aggregation of figures unreliable. Inconsistency can arise from different methods of measurement, interpretations of measurements, or measurements encompassing different entities –

meaning that certain aspects are included in measurement in some cases and excluded in others.

Because of this complexity, accurate and reliable energy consumption measurements are difficult or even impossible to achieve. Some systems may be services offered by third parties, and their energy consumption may be vendor's internal information that is not shared.

Traditionally, processes are improved using measurement data to identify problems. Energy efficiency and greenness in software can and should be improved, even if measurements are lacking or inadequate. We don't have time to wait for improvements in measurement methods; we need to reduce emissions now.

This book provides different mental models that can be used to approach the problem from various angles and make genuine improvements. No single solution will solve every problem, but when application developers have a toolkit for thinking about energy efficiency, improvements can still be made. It is essential to set aside enough time to use this toolkit, which requires prioritising energy efficiency over other matters.

The measurement data will improve in the coming years. Information on the energy consumption or carbon footprint of cloud services can already be obtained from the service administration interface. Processor manufacturers also offer test beds for measuring power consumption at the chip level.

There is still a certain stigma around publishing energy consumption data – it is considered potentially detrimental to the company's reputation. The more organisations publish their measurement data, the easier it will be to compare the energy efficiency of one's own solutions with similar ones. Since we are not there yet, this book focuses more on change than on measurability.

That said, if you have the opportunity to measure energy consumption accurately enough, it is definitely worth using that opportunity and incorporating the metrics into your own production process. This way, you can ensure that a new version of the software does not compromise

energy efficiency. If measurement is not possible, a similar assessment can be made, for example, by monitoring the response times of the service: an extended response time indicates an increase in power demand.

I also strongly recommend publishing as much relevant measurement data as possible, as our understanding of efficiency or its absence is very incomplete due to lack of information.

Summarised

1. The energy consumption of software is practically based on the consumption caused by data processing and transfer. However, the complexity and layering of software make precise measurement challenging.

2. The complexity of measurement is compounded when multiple devices and different abstraction levels are involved. These factors also give rise to commensurability challenges.

3. Despite measurement challenges, improving software energy efficiency is crucial for emissions reduction. This book provides various mental frameworks for addressing this issue.

4. Public measurement data is scarce. Therefore, it is important to aim for publishing one's own measurements to enhance transparency in the field of energy consumption.

5. Accurate measurement of energy consumption should be integrated into the production process whenever possible. If not feasible, alternative approaches such as monitoring response times, can help assess energy efficiency.

5 The Energy Consumption Model of Modern Software

Most of the currently developed software is based on the client-server model, where there is a server on the network that provides various services to an application running on the end-user device. For example, almost all services that are used in a web browser are of this type, as are most mobile applications and an increasing number of desktop applications, where the typical use case is to verify the validity of licences or subscriptions.

The details may vary significantly between different software, but for the purposes of calculating energy consumption, the model can be condensed into three parts:

1. **The energy consumption of software running in the data centre.** This is the location of the actual application server, which contains the centralised business logic of the software and maintains a connection with the application on the end-user device. In addition, there are storage and database servers, as well as possible auxiliary servers, such as authentication and backup solutions.

   In a cloud environment, the boundaries of the application can be less clear, as the code is relying on the cloud environment's ready-made services, which also form an essential part of the software's runtime environment.

The internal data transfer in the server centre is calculated in the same moment.

2. **Data transfer between the data centre and the end-user device.** The energy consumed in data transfer is easiest to think of as linear in relation to the amount of data. The content of the transferred data can vary greatly between different applications, but the energy consumption is the same byte by byte as the data travels the same transfer path. On the other hand, there are significant power differences between the transfer paths.

   Processing the data on the server, such as preparing it for transfer, packing, encrypting, and decrypting, is included in the server centre's energy consumption, and similarly, the same operations on the end-user device are included in the end-user device's consumption.

3. **The application used by the user on the end-user device.** The software running on the end-user device can be very simple and static, a web page, or a complex and intricate application installed on the device.

   For the purposes of calculating energy consumption, it is essentially the same whether the device is running native software specifically written for that purpose or whether the same functionality is implemented as a web application that runs in a browser. In the latter, the energy consumed by the browser is calculated into the consumption of the application.

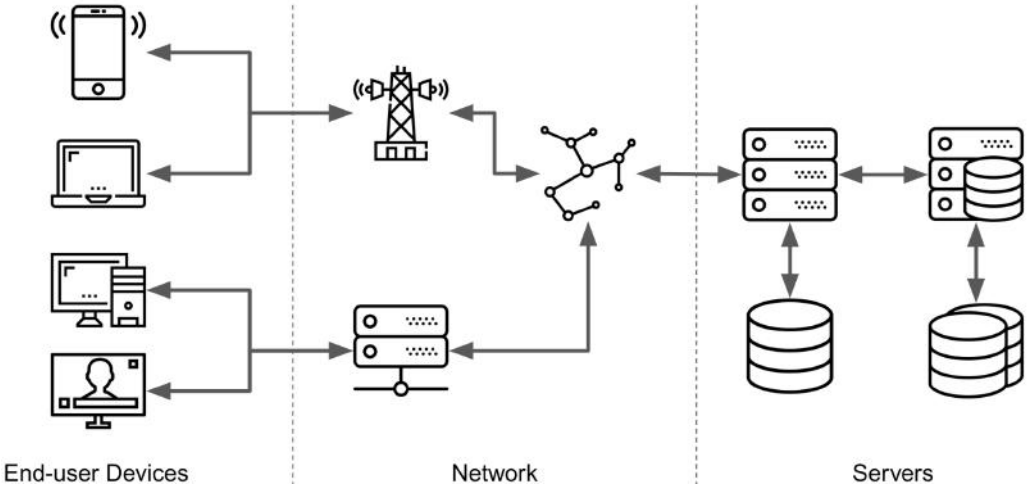


Figure 2. The three main areas of modern software energy consumption.

All IT systems are designed to process and present data to users in an understandable format. Processing can take place in both the data centre and the end-user device, and the location of the processing can significantly affect the amount of data that needs to be transferred and the number of calculations that need to be performed.

Therefore, when designing the architecture of an application, it is important to strive for a balanced solution that minimises both the processing and the transfer, taking into account the specific features of each subsystem in terms of energy consumption. This may sound simple, but in practice, the problem is not trivial. In addition, the planned and actual implementation might be different – as the former Soviet premier Viktor Chernomyrdin famously said, "We wanted the best, but it turned out like always."

It is especially dangerous to focus on optimising only one area at the expense of the other two, unless there is a very compelling reason, such as a narrow communication bandwidth or significant costs, for example, when communicating with space probes. It may easily happen that more power is needed elsewhere, or the user experience deteriorates and users abandon the service.

Let's now examine each of these three areas in more detail, as well as some of the more complex implementations that complicate this simple and clear picture. It is important to keep in mind that these models are all rough generalisations, and the specific features of each piece of software must be taken into account when optimising it. However, these models can help open a discussion and deepen the understanding of the specific characteristics of one's own solutions step by step.

5.1 Data Centres and Cloud Services

The data centre typically hosts the application's centralised functionalities and data storage. The application can be a simple piece of software that runs on a single device, or a complex solution spread across many servers. In terms of energy consumption, every combination is treated the same way: the amount of energy used by the application is the sum of the energy consumed by the servers and the data transfer between them. Some of the servers may be shared among several applications, which complicates the calculation.

In this book, the term "data centre" refers to both physical data centres and servers located in the cloud, which in the end are physically located in one or more data centres. The specific features of the cloud will be highlighted when they differ significantly from traditional server models in terms of energy efficiency.

The organisation developing the application typically does not own the server hardware, but instead rents it from the data centre. Large companies may own their own data centres, but the calculation of energy consumption and emissions remains the same. The production, logistics, maintenance, and decommissioning of equipment generate emissions in addition to the energy consumed by the equipment.

To save energy and reduce emissions, it is essential to minimise the number of servers and to maximise the load on each individual server as close to the ideal level as possible. However, overloading the server does

not make sense because it leads to queuing of requests, longer processing times, and also shortens the life of the server.

It is also worth noting that distributing the service across several physical servers increases the need for other hardware, too. For example, load balancers or shared disk packs are needed, and they consume their share of energy. Decentralisation also complicates the application, as all data processing no longer happens in a shared memory space, and the application must be designed to ensure data consistency.

Modern environments are virtualised or containerised – a private environment is created, configured, and adapted solely for the application. Several such virtual machines or containers are installed on a single physical device, whose capacity is shared among several applications while keeping the environments separate.

These intermediate layers consume a small portion of the capacity of the physical hardware. On the other hand, they enable the physical devices to be used as efficiently as possible and can also offer automatic load balancing. According to a study, container technology does not significantly reduce the efficiency of processing or accessing memory. There is more overhead in data transmission, as virtualisation and containerisation increase processor usage to move data through several layers.[15]

Similarly, the application can be built as a set of microservices: each functionality is a separate service in a separate environment, and the services communicate with each other by using network connections. The advantage of such a solution is scalability – in principle, each microservice can run on any server in the data centre – and the simplicity of individual services. The challenge, on the other hand, is the fragmentation of the software architecture into several smaller parts that also blurs the energy consumption. Microservices allow data centre and cloud service providers to allocate resources in an optimal way, which in turn reduces overall energy consumption.

[15] ieeexplore.ieee.org/document/7095802

5.2 Data Transmission Paths

Data transfer between the data centre and the end-user device is not generally under the control of application developers, at least not all the way through. The data centre is typically connected to the Internet's core network, and that connection is on practical terms already optimised for energy efficiency due to the large amount of transferred data. The choice of data centre, of course, affects the energy consumption of the connection, but a conscious choice can only be made if the consumption information is publicly available. In addition, content delivery network (CDN) solutions can shorten the data transmission path from the consumer to the online content.

The application user, on the other hand, has a great deal of control over the transmission path. They use the service in the location of their choice and with the network connection available at that location. The energy efficiency of the transmission path can be analysed through two essential parameters:

1. **Physical implementation of the transmission path.** Various ways of transferring information differ significantly in their energy consumption per gigabyte. Optical fibre is the most energy-efficient type of network to transfer information, and thus it is used in backbone networks and, for some lucky ones, also as the last link to the home or office. At the other end of the efficiency spectrum is mobile data transmission, which is hundreds of times less efficient than optical fibre.

   According to a study[16] conducted in Finland in 2020, mobile data transmission consumed energy at a rate of about 220 Wh/Gb[17].

[16] joonasnuutinen.fi/wp-content/uploads/2022/01/Nuutinen2021_A-Comparison-of-the-Energy-Consumption-of-Broadband-Data-Transfer-Technologies.pdf

[17] ficom.fi/ajankohtaista/uutiset/digiratkaisuilla-energiatehokkuuteen-mutta-ei-ilman-sahkoa/ (available in Finnish only)

Another study found that the least efficient fixed network in the study consumed 0.25 Wh/Gb, while the most efficient consumed 0.02 Wh/Gb[18], making them a thousand or even ten thousand times more efficient than mobile networks.

2. **The distance of transmitted information.** The longer the distance the data must travel from the data centre to the end-user device, the more energy the transmission consumes – regardless of the transmission path used. Intercontinental transmission networks are fundamentally very efficient. Transferring data from the United States to Europe may consume a fraction of the energy compared to the last kilometre from the base station to the mobile phone.

Also the carbon dioxide emissions of the transmission network energy consumption must be considered. The emissions vary depending on the electricity production method and possible compensations, but typically this information is not publicly available. A good estimate can be obtained from country-specific average estimates; for example, during the six months preceding writing this text, February–July 2023, the electricity consumed in Finland produces 34 gCO2/kWh emissions according to Fingrid[19]. Correspondingly, Germany's average for 2022 is 385 gCO2/kWh[20], meaning the emissions are more than eleven times higher than in Finland.

The transmission routes of global services run through different countries, making it practically impossible to calculate the exact energy consumption of data transmission. In addition, in modern applications and especially in web services, the end-user device typically connects to several background systems – for example, analytics, videos, or chat – and each of them uses different transmission routes.

[18] www.prysmiangroup.com/staticres/energy-consumption-whitepaper/26/index.html

[19] www.fingrid.fi/en/electricity-market-information/real-time-co2-emissions-estimate/

[20] www.statista.com/statistics/1290224/carbon-intensity-power-sector-germany/

Likewise, a server software may be connected to other back-end systems over the network. In these situations, it can safely be assumed that the data is transmitted only on the backbone network.

It is therefore impossible to determine the specific kind of transmission path to the end-user from the data centre. In principle, the type of terminal device can be used to determine the transmission path. But on the other hand, a desktop computer can also be online via mobile broadband or, similarly, a mobile phone can be connected to a wireless local area network and from there to a fibre optic network. A mobile application can determine the quality of the connection used from the operating system of the mobile phone. Similarly, some browsers offer a separate interface for finding this information.

Common to all transmission paths, however, is the effect of the amount of data transferred on energy consumption: the more the application transfers data between the data centre and the terminal device, the more energy is consumed.

In addition, as the amount of data transferred increases, new transfer capacity gets built, which causes emissions. Minimising the transferred data both reduces the energy used by the transmission path and slows down the need to grow the network's transmission capacity.

5.3 End-User Device

Part of the software runs on the end-user's device, either as an independent application or within another application. Typical examples of the first are applications installed on computers and mobile phones, while the second model is represented by web pages and services. It is difficult to draw a clear line between these, as installed applications may display web content as part of their own user interface. While it is challenging to draw the line, it is essential to optimise all parts used and fit them to their purpose.

The data processing of the application can be done both in the data centre and in the end-user device. The location of the processing affects,

sometimes significantly, the amount of data transferred. The chosen architecture should aim to minimise the total energy consumption of all components. There is no universally valid model for this, because the operating models of the applications differ greatly from one another.

In addition, there are significant differences in the energy consumption of devices. For example, mobile phones are optimised to be energy efficient, because their size and weight limit the use of larger batteries and users want to use the device a whole day with a single charge.

Similar optimisation is not necessary for devices connected to a fixed electrical network, such as a 65" flat-screen TV. Of course, the energy classifications of the EU and other similar bodies aim to make devices as energy efficient as possible, but the differences between devices are still significant. It is worth noting that energy classifications have been made for household appliances and televisions[21], but there are no corresponding classifications for information technology devices yet.

For example, watching a 4K video on a large TV and streaming it via a mobile network is a poor solution in terms of energy consumption compared to, for example, watching the same video on a mobile phone or laptop that is connected to a wireless LAN with a fixed connection to the Internet. In the latter case, the video resolution would probably be scaled down automatically due to lower screen resolution – a cleverly implemented application would take this into account, saving both energy and data transfer costs.

The transfer of the application to the terminal should also be considered. If the application is installed on the device, it is transferred only once, and then possibly additional information is transferred from the network only when the application is started for the first time after installation. Each subsequent use of the application requires less data transfer.

If, on the other hand, the application is a web page, it may be transferred to the device again every time it is used. Browsers use caches to store

[21] ec.europa.eu/energy/eepf-labels/label-type/televisions_en

downloaded files on the device to reduce network calls, but the size of the cache is not infinite and at some point the browser cleans out less used files. Similarly, a mobile phone user or the phone itself will automatically delete infrequently used apps to free up storage space, and some of them will be downloaded back when needed again.

Mobile and desktop applications can manage their own caches, allowing developers to maintain precise control over what is transferred. If there is more information to be stored than space can be allocated for the cache, developers should consider a reasonable cache strategy to make space for new files as needed. Once again, the strategy is determined by the application's architecture and usage patterns. On web pages, it is possible to store information in the local storage provided by the browser, but it is managed not only by the application but also by the browser.

The displays of devices are also significant energy consumers. In practice, the display of a mobile phone consumes the most energy in scenarios where it is turned on. Only in the most networking-intensive operations is the display not the biggest consumer.[22] The energy consumption of displays is fairly directly dependent on brightness settings. The ratio of bright and dark pixels also affects consumption in OLED displays, as they do not have a separate backlight. The darker the screen, the less energy is consumed.

The update cycle of devices significantly affects the emissions of the devices. According to device manufacturers, approximately 15-18% of a device's total emissions occur during use and the rest are related to manufacturing and logistics. For example, Dell has comprehensive documentation on the emissions of manufacturing, logistics, use, and disposal of devices.[23]

---

[22] trustworthy.systems/publications/papers/Carroll%3Aphd.pdf

[23] www.dell.com/en-us/dt/corporate/social-impact/advancing-sustainability/sustainable-products-and-services/product-carbon-footprints.htm#tab0=0

5.4 Towards More Complex Models

As stated earlier, the basic model can be – and usually should be – enhanced with additional components designed to either speed up the application or make it more efficient. Such solutions include:

- **Caches** that can store either ready or intermediate processing results to speed up future requests. The cache can be located in front of the service (proxy), in which case it stores responses to requests according to network traffic and serves the corresponding requests directly, or next to the service, in which case the service stores results it often needs in the cache. In some systems, the cache can be an integral part of the implementation, such as in databases.

- **Content Delivery Network (CDN)**, which creates a local copy of online resources close to the user. CDN speeds up the loading of resources and, if the resource is created on the fly (for example, a web page), it also enhances the operation of the application by providing a previously retrieved version of the resource and effectively reducing the energy needed to recreate the resource. CDN can also be used to protect against DDOS attacks. CDN works best in situations with many users or expensive resources that need to be created programmatically.

- **Geographical distribution**, where software executed in data centres is placed in multiple locations. This optimises data transfer and shortens response times. Typically, geographical distribution requires a significant number of users or the response times being business critical.

  If the application is not completely stateless in data centres, the solution requires some level of synchronisation of the data in the centres or redirecting users to their "own" centre. These increase energy consumption compared to operations in a single data centre.

- **VPN connections** change users' transmission paths by routing traffic, for example, from the end-user device to the company's internal network. Further, this network can be geographically distributed and also use VPN connections to secure traffic between its different parts. Every VPN connection involves the encryption and decryption of information, which also increases the need for computing power.

- **Analytics**, which monitor a user's actions on web pages or in the application. Typically, a ready-made service and its tracking library are used. This solution increases the size of the application and creates a kind of side stream of data transfer between the end-user device and the analytics service, which can be more frequent than the service's own data traffic.

With all solutions, one must weigh the advantages and disadvantages, including energy consumption. These solutions often improve the end-user's experience, such as response time, or provide insight into users' actions for developers and other stakeholders in the application development project. But the energy consumption of the solutions is not considered at all.

5.5 Implementation, Testing, and Deployment

Although the majority of software energy consumption occurs during usage, the role of software implementation and testing should not be overlooked. Especially recently, various AI-based programming tools, such as GitHub Copilot[24], have become more prevalent, significantly amplifying the energy consumption associated with software development. Of course, they also expedite programming significantly and, if fortunate, even produce higher-quality code than developers with limited experience.

[24] github.com/features/copilot

Programming itself does not fundamentally differ from other office work; both involve spending much of the workday between a screen and a keyboard – unless one is in a meeting – and the energy consumption of tools is not significantly high.

On the other hand, testing can initiate processes that are multiples larger, usually in an entirely automatic manner. When a programmer commits their code to version control, several different processes are typically initiated to ensure the quality of the produced code. The code might be reviewed using various analysers, and afterwards, a functional version of the software is built, installed on a test server, and subjected to a predetermined number of tests. If errors are detected in this process, the code is returned to the developer for corrections to the identified issues.

The process enhances the software's quality because the aim is to detect as many errors as possible right from the outset. The costs of error correction increase the later the error is discovered. The most expensive correction occurs in software that has already been deployed. According to some estimates, the cost is around 30 times higher for finding a bug during production compared to discovering it during the design phase.[25]

However, this testing process is burdensome and is triggered for each change committed to version control. If the change, for instance, is a correction of a typographical error, the process is entirely unnecessary. Nevertheless, the current systems' ability to differentiate between a correction of a typographical error and the implementation of a new feature remains unfortunately poor. Hence, it is safer to always rerun all the tests.

If an application is compiled separately for multiple – or even several tens of – target systems, the automated system can be highly complex and heavy. The application is compiled anew for each change for each target system. The virtualised environment of the specific system is initiated, the

[25] The exponential cost of fixing bugs, deepsource.com/blog/exponential-cost-of-fixing-bugs

application is installed, and finally, the tests are executed. This is repeated even hundreds of times per day for numerous platforms.

The energy consumption of testing systems, much like other IT energy usage, is not monitored and hence remains in the shadows. This situation should be changed, considering which tests are worth running with each change and which, for example, should be executed once a day or even on longer cycles. Similarly, focusing tests solely on the modified part of the software would be logical, but such integrations are rare. And, naturally, there are instances when modifications to one part of the software can unexpectedly affect functionalities in entirely different areas.

Deployments are also automated in modern environments. This significantly reduces human errors, and the process performed by machines is faster than human-directed processes. However, the frequency of these deployments should be considered. In some organisations, deployments are done multiple times a day, while in others, they occur weekly or monthly. Of course, the nature of the software affects this, but too frequent deployments can easily lead to unnecessary energy consumption.

Summarised

1. Most modern software follows a client-server model where servers provide services to end-user devices. The model's energy consumption is divided into three parts: software in data centres, data transmission between data centres and devices, and the application on the end-user device.

2. Energy usage in data centres includes application servers, storage, and databases. Cloud environments and virtualisation impact energy consumption. Optimising server usage and load is key to reducing energy consumption.

Summarised

3 The energy efficiency of data transmission depends on the route and distance. Network type – such as mobile data or fibre-optic – and the distance data is transmitted affect consumption. Minimising data transmission reduces energy use.

4 Energy consumption in end-user devices varies based on device types – like mobile phones, desktop computers – and usage scenarios. Screen brightness, caching, and data transfer methods affect consumption. Device upgrade cycles also impact emissions.

5 Additional components like caches, content delivery networks (CDNs), geographic distribution, VPN connections, and analytics can significantly influence energy consumption.

6 Raiders of the Lost Efficiency

Back in the day, computers were very inefficient and had only limited memory or storage space. Programs had to be efficient because otherwise they could not be implemented or executed at all. Nowadays, there is practically unlimited power available, at least in relation to day-to-day use, and thus it can be easily wasted. Only specialised use, such as scientific calculation, training artificial intelligence, or handling large amounts of data, forces optimisation to be an important part of software development.

Software has also become layered. At the dawn of computing, all code had to be written by hand, and gradually libraries began to appear for implementing commonly needed routines. Now there are libraries or ready-made routines available for most tasks. It makes sense to build software in layers with libraries to ensure quality and development efficiency, as applications and their runtime environments have become significantly more complex and rich in features.

However, this can easily lead to intellectual laziness. It is easier and less expensive to choose a ready-made library than to write a routine yourself, even if the routine is fairly simple. Of course, the solution provided by a library is probably field tested and therefore more reliable, but the code inside of a ready-made library is rarely looked into. If the solution is fast enough – albeit inefficient on a large scale – no problems are noticed during development.

This is not to say that every software developer should write all their code from scratch to be sure of its energy efficiency. This is inefficient both in terms of developer productivity and surprisingly often with energy efficiency because the code of frequently used libraries becomes gradually more optimised through practical experience.

If the code written to replace a library becomes hundreds or thousands of lines long, the chances of errors increase significantly. Similarly, problems typically arise if the code has to perform complex operations or there are a significant number of possible deviations from the normal process. These situations should raise alarms, and solutions should be analysed calmly, preferably with the help of a colleague.

Reassessing one's ways of working always makes sense to figure out whether the problem could be solved in a different manner and more efficiently. When these solutions are found, they should be shared among other developers.

6.1 Waste

Similar examples can be found in various areas of software development. On the other hand, each application is unique with its own strengths and weaknesses. To approach the issue more analytically, it can be structured using the concept of waste, which is familiar from lean manufacturing.

“Lean is founded on the concept of continuous and incremental improvements on product and process while eliminating redundant activities.”[26]

In terms of energy efficiency, waste can be defined as extra, unproductive activities that consume energy unnecessarily. Like in lean, there are different types of energy waste. In the following, waste is discussed mainly from the perspective of companies, because most software is at least partially owned or used by companies.

[26] en.wikipedia.org/wiki/Lean_manufacturing

This discussion assumes that the application itself has been implemented correctly. If an application makes a thousand unnecessary database queries in a poorly written loop in each of its operations, I consider it primarily a programming error and not just waste – even though that operation in itself is wasteful. Waste is more laziness in thinking or implementation than actual mistakes, and eliminating waste is very rarely straightforward.

There are many different types of wastes – the list below is by no means complete or final – and their impact on energy efficiency varies significantly depending on the nature of the software, how it is used, and the overall architecture around it. The types of waste are listed in order of significance according to my own thinking, which means that on average, the biggest changes can be made by addressing the waste that appears higher on the list. However, as mentioned, this is specific to the software in question and the main goal of the list is to offer different perspectives on improving efficiency.

6.1.1 Redundant Software

If the software as a whole is useless, all the energy it uses is wasted. Of course, defining the usefulness or futility of software is very subjective. Typical examples of clearly pointless applications are remnants from earlier times. For example, a system that monitors an inactive process.

Another example is pre-installed applications that come with consumer computers, many of which, in my opinion, are only important for the device manufacturer and a complete waste for the end-user.

If the software is found to be a complete waste, it should be taken out of use immediately and the resources it uses should be saved for the benefit of the planet.

One way to find unnecessary software is to go through all the ICT suppliers with the financial department and find out which services are bought from them and for what need. If the corporate ICT architecture is not documented and maintained, then such a project can generate clear

savings by removing unnecessary solutions and adjusting the running environments of the solutions used.

6.1.2 Improper Use

Software lives in time, just like everything else in companies. When the situation of a company changes, the software may be left not updated and its use may no longer serve its purpose. Or it was not originally implemented or purchased in a suitable manner.

In particular, ready-made software and SaaS services can be used partially or even entirely for different purposes than they were originally designed for. This does not mean writing a novel in PowerPoint, but rather managing the recruitment process with a CRM designed for sales needs. Both processes are similar and such cross-use may work well.

Difficulties arise if the application does not support all stages or features of the supported process, or does not allow customisations to meet them. This can result in a situation where, for example, heavy searches are constantly needed to find certain information. This is usually also visible to users as slowness of the application or a particular function.

Typically, such a solution has been tried and found to work well enough. Then the needs are refined – for example, through practical experiences – and the inflexibilities of the solution become challenges or obstacles. At this stage, the application may already have stored data for a year or two, which is then difficult to migrate to a better system. Similarly, the implementation or procurement costs of a better system may be too high in relation to the achieved benefit, or the sunk costs fallacy[27] may mislead rational decision-making.

Similarly, the solution may start loafing as it gradually changes over the years – the software's various modifications form historical layers in the application. If it is a solution produced by a company, the application developers also change between versions and there is no time or desire to

[27] en.wikipedia.org/wiki/Escalation_of_commitment

study the previous implementations in an unhurried pace, but new code is just implemented next to or on top of the old code without understanding the system as a whole. In this case, the functionality of the application may meet its purpose, but its implementation is not necessarily optimal.

Removing historical layers is difficult because they are typically not clearly separated from each other, as simultaneously new code has been written and old code has been modified to work with the new. Refactoring the application completely or partially would solve this problem, but it is expensive and error-prone. Still, there are situations where it is the most cost-effective path forward. A careful assessment and decisions based on it are key when considering refactoring. Good test coverage helps to detect any errors that may occur during refactoring.

6.1.3 User Mistakes

"People make mistakes, computers automate them." Anon.

Application users make mistakes. The finger or mouse cursor is in the wrong place, typing errors occur in input fields, and so forth. Every mistake triggers a series of actions that are waste from the application's point of view. Similarly, the actions needed to prevent errors are also waste, but they are useful in preventing larger waste.

The better the application prevents the user from making mistakes, the less it needs to do unnecessary actions. Similarly, the less effort the user needs to put in to complete their task – in terms of the number of screens or inputs – the less the machine needs to work. Such matters are more related to the user experience and interface than to code efficiency, so other experts besides software developers are needed in solving them.

Users are different from each other in their premises and they may have limitations in using the application. Fortunately, ageing does level the differences between people by degrading everyone's cognitive and physical abilities. Taking these matters into account is called accessibility

and it has been on the rise lately due to the EU's accessibility directive[28]. Accessibility is also energy efficient because it reduces errors by making the application easier to use for everyone – not just for special groups.

It is worth noting that removing errors does not necessarily mean better testing or instructions, but the best result is achieved by eliminating the possibility of error. A bigger and almost zen-like question is how this is done in practice.

Eliminating or reducing the possibilities of errors is holistic design work that ponders on the process and the application facilitating it. The process is made as straightforward as possible and data already known by the application is used to the full benefit of the user. The less interactivity the user is required to have with the application, the less there is danger of human errors.

On the other hand, errors can also be removed by simple user interface or text changes. User interface testing is a key activity, as the software developers know the system too well and thus are not the best people to consider how the system is actually used in real life. Similarly, reading user feedback and interviewing people working in customer service may be an eye opening experience to find the real challenges. These channels do not provide ready solutions, but formulating the problem eases making the changes.

6.1.4 Wrong Architecture

The architecture of an application determines what is easy and what is difficult to implement in the application. The architecture is not always optimal considering the tasks of the application, and the code of the application may have to "fight" against the architecture or features that need to be implemented in an unnecessarily complex way. This increases energy consumption and also the likelihood of errors in the software.

[28] eur-lex.europa.eu/legal-content/EN/TXT/HTML/?uri=CELEX:32016L2102&from=EN

The typical reason for waste caused by the architecture is its ageing as the purpose or usage of the application changes over the years. Sometimes the chosen architecture is wrong to begin with, but this should become evident during the development of the first version and should be reacted to. The library or application framework used by the application may also be wrong or difficult from an architectural point of view, leading to various needs for adaptations and other energy- consuming difficulties.

Changing the architecture means practically rewriting part of the application, so it is not a small operation. If the structure of the application is modular or it is implemented as independent solutions, they may be adopted with little effort in the new architecture. But the worst scenario may be rewriting the entire application.

The architecture can also be extended to the enterprise architecture of a company that defines the interaction and compatibility of different systems related to running the business. This can also be built improperly in relation to business objectives, causing waste in the processing, transfer, and storage of information. Changing this higher level architecture is an even bigger task than modifying the architecture of a single application.

6.1.5 Wrong Data Models

The main task of software is to process information and present it to the user in a form that is understandable to humans. Information is stored and processed within various data models that try to organise the information according to the operation of the application and the needs of people.

This is not always successful, and the data model may be incomplete or inefficient for the intended purpose. Incomplete data models were a major problem when memory and storage space were limited and data models were often designed based on their required number of bytes. If some information was not available, it could either not be processed at all or it was squeezed into a less necessary field of the data model. These problems have been solved with the increase in memory capacity and nowadays it is relatively straightforward to expand the data model within the application, unless the application is embedded software. However, if

the data model is derived from background systems or is otherwise integrated into a larger whole, refactoring it can be difficult.

The greater challenge is to choose the right data model for the need. There is a risk that the desired use of the application is not aligned with the used data models, and thus the code has to constantly go through the data structures searching or combining records to display to the user.

Fixing such a problem is usually unfortunately challenging, because the data model, together with the architecture, determines the internal structure of the entire application. The necessary changes are deep and error-prone. Often, the problem is tolerated instead of being solved.

One subtype of data model waste is the incompatibility of the structures of used databases. Although database engines are excellent at optimising queries, they cannot, in the end, fix structural problems. Refactoring database structures is not straightforward, but especially for solutions that serve heavy loads, the effort pays off.

Regardless of the modifiability of the database structure, it is important to ensure the existence of the correct indexes. Indexes by no means solve all problems, but they are still worth using.

There are also differences between database engines, for example, relational and document databases can be used to solve certain kinds of problems efficiently. On the other hand, they can be extremely inefficient when used against their designed purpose.

6.1.6 Redundant Data

Very few applications actively delete information. There are clear cases where user information is deleted when use ends or when GDPR[29] requires it, but software generally – and unnecessarily – likes to hoard all sorts of information.

[29] General Data Protection Regulation, en.wikipedia.org/wiki/General_Data_Protection_Regulation

Data models are usually designed from the perspective of data integrity and searchability. Deletion is not necessarily considered, because the data may be needed for years or even forever. But not all historical data is equally valuable, and it is a good practice to also consider the data model from the perspective of deletion. For example, it is probably not reasonable to keep lists of tasks from ten years ago, but they will not disappear from the service by themselves and users cannot be trusted to delete all unnecessary items.

In the design phase of applications, I recommend designing the data model to allow partial deletion of data. This is smart both from the perspective of GDPR and energy efficiency. Unnecessary data, after all, multiplies: its effects show up in longer processing operations in databases, in the use of disk space, in the duration of backups, and the size of copies, potentially in a separate database of an external search engine, and in its backup, if one is taken. The list is long and cumulative.

It is a good idea to occasionally examine the contents of the data stores of older applications. You may find completely unnecessary data that is not displayed or used anywhere, but is just wasting space in the database. Or it is displayed and processed, but the users do not need it for anything – doubling the waste.

6.1.7 Non-optimised Data

Modern applications transfer a lot of data both internally and with other applications. Since most applications operate on a client-server architecture, data transfer is an essential part of the application's operation.

As previously noted, data transfer has an energy cost, which is based on the amount of data transferred. In principle, the less data is transferred, the less energy is used.

The amount of data transferred can be controlled by adjusting the type of data transferred and how often it is transferred, in other words changing the frequency of the transfer. Since data transfer is free or very cheap for

app developers – unless there are huge amounts of data or users – it is rarely given much attention. The typical situation is that users complain about application slowness or the data transfer bill that surprises the software owner.

Typically, two kinds of data are transferred:

1. Communication between application components, which is determined by the protocol between the parties.
2. Various files or streams, such as images or videos to be presented on the client device or attachments to be stored on the server.

It is more difficult to affect the volumes of the first category, for instance, the conversation between the application and the database is conducted using set protocols. Of course, the data transfer volume can be affected by trimming the data retrieved or sent, but this is not always possible.

The second category can be affected with user experience design by reducing the amount of files displayed or replacing the presentation with a less consuming one. For example, changing the used file format to a more efficient one, or reducing the resolution or number of colours for images and videos.

Efficient caching of files on the client device or transferring them as part of application installation can reduce the need for data transfer. Using caches increases the application complexity and makes development more expensive. In any case, it makes sense to reduce the file size and it is worth compressing everything possible.

There are various services on the Internet that can optimise files. For example, significant reductions can be made in the size of images or fonts. The book "Sustainable Web Design"[30] introduces and lists a large number of solutions with practical examples. For example, significant savings can be achieved for fonts by removing unnecessary variants and glyphs that are not needed for the application or website. In the removal of glyphs, it

[30] abookapart.com/products/sustainable-web-design

is important to consider future safety: even if a certain glyph is not needed at the moment, it may be needed in the future, for example, to correctly display the name of a customer whose name might contain missing glyphs.

6.1.8 Redundant Data Transfers

It is often easier to transfer data multiple times for the sake of certainty or convenience. This is easily overlooked because transfer networks are fast and the energy consumption is hidden. For example, in data synchronisation, the entire bundle of data may be transferred from the server to the client device every time instead of just changes. The reason for this is fear of synchronisation failure or data degradation over time due to uncontrolled changes. The problem is not easy to solve, so avoiding it is attractive.

Various synchronisation libraries and protocols have been developed to solve the problem. Using these can result in significant savings in data transfer. It should be noted that the implementation of synchronisation can make the development of the application, and especially the search for errors, more complicated. There is also a risk that the data will remain only partially coherent. These risks should be considered in relation to the benefits achieved.

There are also truly decentralised protocols, in which each device both sends and receives data, so that data can be obtained from a location closer to the client than from the server.

6.1.9 Algorithmic Inefficiency

Finally, we come to a topic that is typically the first to come to mind for an experienced software developer when talking about efficient coding. The efficiency and, above all, suitability of the selected algorithms and data structures for the task have traditionally been major pain points in evaluating the efficiency of software. The topic is taught in university courses and working with algorithms is usually enjoyable for developers.

The algorithms and structures of long-used programming languages and libraries are usually thoroughly optimised and therefore energy efficient as long as they are used for the right purpose. On the other hand, the efficiency of self-written or incorrectly chosen algorithms can be very poor – especially, if the developer does not have the time or ability to consider the chosen solutions from an efficiency perspective.

The most efficient algorithms and data structures are often also more challenging to understand than basic algorithms that solve the same problem. It is also important to understand that old and once efficient algorithms may not be efficient on modern processor architectures.[31]

Typically, there are certain parts of an application whose code is executed significantly more than other parts of the application. These include, for example, the handling of events performed by users (event loop), an ORM library maintaining database connections, or a component monitoring user sessions. It is worth identifying such hubs of activity through profiling, or if that is not possible, through architecture or data flow analysis.

It makes sense to invest more in optimising hubs and leave less used parts as they are. Sometimes such a hub is in the library or application framework code and optimising it may be impossible. In this case, the only available action is to find the next candidate for optimisation until your own code is reached and work can begin. Otherwise, the significance of the targets becomes too small.

6.1.10 Misguiding Users

In addition to users making mistakes, users are often misled in services, or achieving certain goals is made difficult, for example, closing their account or requesting help. Dark patterns are user interface solutions that deceive

[31] jolynch.github.io/posts/use_fast_data_algorithms/

the user into acting against their own interests because it benefits the party offering the service.[32]

Besides active misleading, for example, trying to convince the user to purchase a certain service with a high price, there is also passive misleading. In such cases, the user is prevented from finding a functionality that is negative for the company, or using it has been made difficult. A typical example is hiding the cancellation of an order in an atypical or semi-hidden location or requesting an order number that is not requested elsewhere in the application.

These misleading practices cause the user to unnecessarily navigate across the service, which increases energy consumption. Similarly, if the user accidentally takes a service into use without actually intending to do so, the ordering and cancellation of that service is a waste.

These design models are harmful in many ways and their use is also ethically questionable.

Misleading solutions can also be implemented accidentally. Even if the idea is good, the implementation may still be misleading to the user. Some business objectives, such as the time spent on the site or the number of pages visited, may lead application development to adopt such misleading models or practices that serve only the business instead of the user.

These models usually surface during user research. Testers do mention if, for example, logging out of the service or ending an order has been made too difficult.

Similarly, UX writing – designing and writing texts related to user experience – helps against accidentally implemented harmful models. For example, it is good to pay attention to the content and formatting of error messages so that the application user does not try again and again to do the same action that cannot actually be done.

[32] en.wikipedia.org/wiki/Dark_pattern

6.1.11 Too Much Code

Nowadays, applications are assembled from a large number of libraries, which provide functionality, data structures, and algorithms that are combined according to the logic of the application. Libraries rely on other libraries, which in turn rely on others. In the worst case, the application's own code may be only a few percent compared to the amount of code in the libraries, and the application may use only a few percent of the libraries' code.

This creates a problem especially when transferring the application to an end-user device. Practically, all applications are downloaded from the internet, and as a result, the same library may be on the device dozens of times within different applications. Yet, only a fraction of the library's code may be used. To solve this problem, it is advisable to use application bundlers that remove unnecessary code and squeeze the application into a tight package.

Mobile and desktop applications are downloaded once upon installation and possibly during updates. Web applications, on the other hand, are downloaded much more frequently, depending on the settings of the cache and browser, which makes the problem significantly larger. This challenge can be reduced by adjusting the file cache policy.

It is advisable to investigate the size and dependencies of a library before deciding to use it in an application. If only a small amount of code is needed, it is worth considering either using a smaller library – if such an option is available – or writing the required code yourself. In the limits allowed by their licences, it may also be possible to copy code from open source libraries into your own application.

The more libraries an application uses, the more often they are updated, but their update schedules are of course not linked to each other. Unfortunately, in newer languages or rapidly developing environments, such as currently in web front-end implementations, libraries are developed almost at a break-neck speed and changes can break backward compatibility. Each added library also increases the maintenance workload.

Of course, there is a flip side to the matter. The code of a ready-made library is typically used in thousands of projects and thus tested thoroughly. The library has a maintainer that updates the library, for example, when security breaches are found. Overall, the library develops further without investing your own time and resources. Of course, the changes might not always have a positive impact, as they may break the functionality of the application or force big internal changes.

6.1.12 Inefficient Programming Language

There are significant differences in the efficiency of programming languages. Some languages are compiled into machine code, some into interpreted bytecode, and some are compiled into bytecode at runtime and then interpreted. In addition, the different emphasis and internal data structures of languages affect efficiency.

However, the choice of used language is not always in the hands of programmers and cannot be justified solely on efficiency – otherwise all software would be developed in machine language, C, or similar languages close to the hardware. The characteristics of a language help and protect the developer in various ways, and over time, certain kinds of solutions have emerged in the ecosystem surrounding them. Although in theory, languages are equal as they are Turing-complete[33], this is not the case in practice. The characteristics of languages, their surrounding ecosystems with libraries and components, and the availability of developers significantly affect the usability of the language in the desired environment and solving the problem in question.

Researchers at the University of Beira Interior in Portugal have studied the efficiency of about twenty programming languages[34] using ten different test tasks. The time taken to complete the task and the memory

[33] If a programming language is Turing-complete, it can simulate the operation of any other computer or program; en.wikipedia.org/wiki/Turing_machine

[34] haslab.github.io/SAFER/scp21.pdf

used have been measured, and the languages have been ranked in terms of efficiency based on these measurements.

The test setup is not perfect, as the used implementations of routines between different languages are not mutually comparable, but rather have been developed by various developers. For this reason, the results can only be considered indicative at best. As expected, the tests revealed significant differences, for example, between compiled and interpreted languages.

According to the study, the three most energy-efficient languages are C, Rust, and C++. Among the more widely used languages, Java, Swift, Go, and JavaScript also performed well. The least efficient languages were Perl, Python, and Ruby, and among the most frequently used languages, Lua and PHP were at the lower spectrum of efficiency. The difference between the most efficient and the least efficient language was almost 80-fold.

The tests were of a mathematical nature, so, for example, the IO capabilities or the efficiency of libraries available in the ecosystem was not examined. These have a significant importance in modern application development, as the current paradigm is based on transferring data and building applications from ready-made libraries.

There are also efficiency differences between language runtime environments. For the same interpreted or bytecode-compiled language, there can be different interpreters, some of which produce more optimised code than others. The efficiency of a language can also increase – or sometimes even decrease – with the new version of the runtime environment without any application code changes. Therefore, it is advisable to ensure that the application is run in the most efficient runtime environment possible.

In the tests, the correlation between execution time and energy consumption was also examined for each language. The correlation was either strong or very strong for all languages. This suggests that, regardless of the implementation language, a faster application is more energy

efficient than a slower one. No similar correlations between memory and energy consumption or execution time and memory consumption were found except in a few languages.

Since software developers may not have the ability to freely choose the language, it is better to focus on optimising the code written in the current language instead of changing languages. Of course, it is possible to write frequently used parts of the system in a more efficient language.

When adding new components to the system, it makes also sense to consider energy consumption in the choice of the implementation language. However, it is not advisable to focus only on energy. There might be a shortage of experts in a more efficient language, there may be no necessary libraries available, or the current and planned languages may not work well together.

However, it is good for software developers to master several languages, as this broadens the project team's ability to choose the language according to the task. Being proficient in multiple languages makes it easier to learn new languages, too.

6.1.13 Waste in Starting a Software

There are basically two types of application environments in data centres and cloud services: the application starts once and serves users until the process is shut down, or the application starts every time to serve a user's request.

If the application is of the first type, it is useful to prepare the application for long-term operation when it starts and do those tasks in advance that should be done in any case at some point during the application's use. These tasks are usually reading settings, opening database and other connections, reading generally needed information into memory, or starting possible subprocesses. If the preparatory tasks are particularly heavy or are needed only in rare situations, it may be reasonable to do them only when the actual need arises.

If the application is of the second type – such as PHP applications or cloud functions – only those tasks that are essential for serving all requests should be done when starting. Then the application should proceed based on what is needed to serve user requests – in other words, the starting process would be divided into several phases. This solution does not fit all situations; the architecture of the application may not allow the scattering of starting phases within other code or the codebase becomes too confusing in terms of further development and error sensitivity.

It should also be noted that the statelessness and fast startup of the application to serve user requests may allow using cold standby instead of hot standby. An inactive copy of the application is not kept running waiting for the active application to fail, but a new copy is started when a failure occurs. This approach saves server resources for other needs.

6.1.14 Redundant Redundancies

In modern applications and websites, there is additional "eye candy" that is intended to make the use of the application more comfortable, more human, or more fun. If these actions do not facilitate the actions of users or reduce errors, they are likely to be waste. This does not of course mean that all applications should be stripped down as much as possible. Eye candy has its place, but it should be curated according to user needs and not scattered all over the place.

We humans are still animals with hunter-gatherer instincts and everything that moves catches our attention. When using software, movement in the user interface very rarely means danger, but our eyes are still naturally drawn to animations. If you want to focus the user's attention on a changed area or a new functionality that has become available, using animation makes sense. On the other hand, there are a lot of cases in which attention-seeking has no reasonable purpose; for example, in a large number of websites elements appearing on the page are animated for show-off purposes only.

It is important that the application tells the user that their press has been registered or the mouse cursor is in an area where clicking accomplishes something. This reduces cognitive load, uncertainty, and errors. Still, it is reasonable to remove unnecessary attention grabbers. If you do not want to attract the user's attention to an animation on the screen, the animation is waste in two ways: it takes attention away from the essentials and unnecessarily consumes energy.

Most background videos can also be added to this category. In general, videos should be used sparingly because they are heavy to transfer and their information value compared to the amount of data is meagre. Videos have their time and place, as they facilitate understanding of matters, but often they could be replaced with animations – which is also usually video but compresses much more efficiently – or a short text.

Technically, some of the performance impact of animations and other similar elements can be mitigated by modifying the behaviour of the application or website. If the application or browser is in the background, in other words another application is on top, the amount of processing should be reduced and all user interface updates should be turned off. On smartphones, the operating system also adjusts the behaviour of applications in this regard.

In addition, some of the application or website functionality may be hidden from the user, waiting at the bottom of the page or the user has already passed the point on the page while scrolling down. In this case, for example, loaded images and videos can be removed to save memory and, if necessary, restored when the user returns. Of course, it must be assessed or, if possible, monitored how many users pass through the views in reverse, so that removal and re-loading does not cause additional burden.

Formats also affect energy consumption, for example, GIF animations play in browsers even when they are not visible, while CSS-based animations automatically turn off when they are off the visible area.[35]

[35] webkit.org/blog/8970/how-web-content-can-affect-power-usage/

6.2 Minimisation

Optimising the energy efficiency of applications can also be approached through minimisation instead of waste. In this case, the aim is not to identify issues or events where energy is wasted, but to create the most compact solution that meets the requirements.

Minimisation requires good advance planning. Each requirement should be questioned and one must consider whether the application can suffice without it. But truly important requirements should still be implemented, as if the result of minimisation is only a half-finished application that needs to be fixed later, the probability of inefficiency or waste increases quickly. On the other hand, a well-planned minimal solution is typically elegant to implement and straightforward to use. In order to succeed in minimisation, both the business and user needs must be known in sufficient detail.

Minimisation works particularly well in situations where the challenges to be solved are clearly defined. In addition, them being numerically measurable helps to understand magnitudes and implementation decisions are not made based on personal biases and preoccupations.

When the main purpose of the service is the focal point of the design, using it is straightforward and the user experience improves. At the same time, conversions increase and user paths shorten, which improves business results, energy efficiency, and user experience. It takes courage to break away from functionality-centricity, but the results are rewarding.

In addition to streamlining needs, narrowing the target group also facilitates minimisation, as it eliminates different usage scenarios. The narrowing is advisable to be role-based, for example, whether to offer views to managers and employees in a service aimed at payroll clerks, or to meet their needs through something else. Expert users and other pro users can also be provided with their own interfaces around the main focus of the application, so that they do not interfere with satisfying the needs of normal users.

In consumer services, it is extremely important to ensure that the narrowing of the target group does not restrict the opportunities of some groups to use the service, as this may lead to discrimination. For example, removing accessibility from the requirements of an application is not sensible minimisation.

The advantage of minimisation is the simplification of the application, with shorter user paths, fewer functions and their error checks, a clearer user interface that is less error-prone, and a lower cognitive load on the user, as the application offers fewer visual messages.

It is also worth remembering that the minimum viable product (MVP) and minimisation are two very different things. MVP is the first version or prototype of a solution, and it is used to test the feasibility of the idea. A minimal application, on the other hand, is a complete solution that is not intended to be expanded later without significant reason.

However, the MVP mindset has the advantage that the application is implemented to meet the assumedly most important objectives. Along the way, more is learned about the markets and users and an attempt is made to make the most sensible decisions based on tight prioritisation. If this does not lead to extra layering or other waste, the result is a tightly packaged solution with no excess functionality.

Summarised

1. The abundant resources available in current software development practices can lead to neglecting optimisation, resulting in inefficiency. Developers should refrain from solely streamlining their own work and should prioritise energy efficiency.

Summarised

2 The challenge of software inefficiency can be approached through the familiar concept of "lean," focusing attention on non-value-adding activities that increase energy consumption.

3 Choosing the right architecture, data models, algorithms, and programming languages, along with managing data transmission and eliminating unnecessary elements, significantly affect software energy consumption.

4 User-centric design and accessible user interfaces minimise user errors and the resulting waste.

5 Minimisation offers an alternative way to build energy efficiency – focusing on the essentials and eliminating excess functions simplifies the software, enhancing user experience and energy efficiency.

7 Solutions

The previous chapter listed a large number of different performance bottlenecks. Building on them, this chapter discusses various ways to reduce the software's environmental load. These solutions generally improve the performance of the software in several metrics, as energy consumption and the execution time of the software are strongly correlated. The time is also reflected in the slowness of the application, which is one of the biggest sources of user irritation.

Similarly, reducing energy consumption also reduces hosting expenses. Changes can be significant in cloud services, where billing is based on usage, clock cycles, or data transferred.

None of these solutions are silver bullets, and they should be applied in the context of your own software. And if all of these are already familiar and in use, congratulations on your ability to produce efficient software. I am also happy to receive new ideas for increasing eco-efficiency.

7.1 Minimise Stored Data

The less an application processes and stores data, the less it also transfers it between the device and the server.

- **Minimising the application data model** – remove unnecessary or outdated information, design processes to easily remove data, store only the final results of the process and destroy intermediate

results. Store data in a suitable format wherever possible, using the data types offered by the language and database reasonably.

- **Minimising the static data and files** – choose the right format and remove unnecessary or unnoticeable items. Adjusting the resolution and aggressiveness of compression of images and videos correctly is essential. It is also worth considering whether the representation of an item can be changed to another form, such as an animation or a pair of images that compress better than a captured video. All of these savings accumulate because the files are processed only once. Some of the technical changes may require manual work. All other tasks should be automated actions as part of the CI / CD pipeline.

- **Minimising user input** – if the user can attach their own images or videos to the service, it is advisable to crop and resize the stored versions into a suitable size, compress them using reasonable algorithm and settings, and store them in an efficient format. These actions should be done on the end-user device if possible. The data coming from the user to the application can also be minimised by limiting the size of the user's inputs, such as the length of a video.

- **Cold storage of data** – sometimes data cannot be removed, but it must be accessible at least with some effort. In these scenarios, cold storage services such as Amazon Glacier or Google Cloud Archive can be a more efficient solution to store data than keeping it constantly available as part of the application.

- **Minimising and deleting analytics data** – applications and websites store analytics data on their own activity, which is used to both improve the technical performance of the service and understand the needs of users. The error-free operation and continuity of the business are both important, but it is worth weighing what data is needed and for how long.

7.2 Minimise Transferred Data

Transferred data is tightly intertwined with minimising data, in other words, the less data is processed, the less is transferred. But in addition, there are special considerations for transferring data:

- **Adjusting frequencies** – determine the most reasonable frequency for transferring data in terms of user experience and the internal state of the application. From the point of view of energy efficiency, the longest intervals without communication are good, but they can deteriorate the user experience.

- **Compressing data** – if the used format does not offer compression itself, it is practically reasonable to compress and decompress all data. Text data compresses to a fraction of the original quite efficiently. Compression gives additional benefit if the data has been combined as described in the item below.

- **Choosing the protocol or message format** – use an energy-efficient protocol or message format. There are differences, as some of them are very chatty and others accomplish the same tasks with a smaller amount of data. If the data is very structured or there is a lot of it, it is worth considering using binary messaging instead of text-based communication. Changing communications from one form to another is always costly, so it is advisable to weigh the benefits of reducing data transfer against the increased complexity of the application and its runtime.

- **Removing presentation data transfer** – the server may produce HTML code that is displayed on the device either in a browser or within the application. If there is a lot of HTML to transfer, it may make perfect sense to transfer only the textual content in a very concise format and render the HTML code on the device using a template library.

- **Transferring only changed data** – if the same data is both on the server and in the application, it is sufficient to transfer only the changed data (delta) to update the other end. There are libraries

available for this, which should be used wherever possible. The change is not trivial and can complicate maintenance and troubleshooting significantly.

- **Identifying unchanged data** – if data does not change or changes very rarely, it can be kept on the user's device without an expiration date. Servers can serve such data from various caches instead of the application server. The data transfer can also be implemented in such a way that the server informs the client application about changed data among other communications and the client fetches the data only after being informed.

- **Verifying data before sending** – the data entered on the user's device should be verified before being sent to the server, which reduces the number of potential error messages sent. It should be remembered that checking on the user's device does not eliminate the need to also check the data on the server. If the check is intensive and the amount of data being transferred is small, it may be environmentally more efficient to only check on the server.

- **Combining data for transfer** – multiple messages can be bundled into the same transmission, for example, reporting user analytics data along with the user's data sent to the server. This requires that the messages are easy to bundle and the result is more compact than the messages separately.

- **Minimising HTTP headers** – most communication today happens over HTTP. Ready-made libraries may automatically attach various headers to requests or responses. The headers should be reviewed and unnecessary ones removed in order to reduce energy consumption.

- **Reducing HTTP redirects** – to make life easier for users, web servers can add redirects when resources move to a new location, so that an HTTP request is responded to by directing to make a new HTTP request to another address. Typical example is a slash added to certain addresses in WordPress. Such extra hops should

be removed whenever possible by using the correct address in the request.

- **Minimising server-to-server transfer** – typically, a database server or similar is a part of the server-side platform, to store the processed data. Minimise the amount of data retrieved in a single request to the right size so that unnecessary data is not retrieved and, on the other hand, a new request is not needed.

7.3 Reduce Code

Reducing the code of an application leads to a decrease in the size of the application, making it lighter to transfer over the network and potentially faster and more efficient to launch.

- **Removing dead code** – if a certain part of the application is no longer used, the code can be simply removed. If the need to return to it arises, the old code can be found in version control. If version control is not in use, it is advisable to start using it immediately. Version control requires both storage space and processing capacity, but in turn significantly improves the reliability of the software development process and reduces errors in the produced solution.

- **Reducing the number of libraries** – most of the application code resides likely in third-party libraries. Some of these are small and efficient, while others are large and poorly implemented. It is difficult to see the difference from the outside. If only a small portion of a library is used, consider the possibility of incorporating the used features into your own application and removing the library. Of course, you need to check whether the library licence terms allow this. Remember that the responsibility for any problems in the code will shift to you.

  Some libraries have a lighter version that contain, for example, 90% of the functionality with much less code.

- **Reducing the features of the application** – if a feature is not widely used or is outdated, it is advisable to remove it from the user interface and then remove the resulting dead code.
- **Compiling the application separately to different environments** – if an application can be installed in a variety of different environments, such as several versions of the operating system, there may be environment-specific parts or two versions of the code for two different architectures in the application binary file. Instead of offering a single file, consider compiling the application separately for different environments and offer the correct version during the application download process. Note that mobile app stores do this automatically.

7.4 Improve Application Efficiency

As the runtime and energy consumption of an application are strongly correlated, any measures taken to improve the efficiency of the application will also improve its ecological sustainability. There is a vast amount of literature on this subject and university-level research is being conducted, so only the surface has been touched here.

- **Choosing the right algorithms** – algorithms have differences, and some are well suited for handling large masses of data, while others are efficient only with small amounts of data. Because software developers are usually in a hurry due to customer or schedule pressure, the first suitable or functioning algorithm may be taken into use without further consideration of efficiency.

  The problem hides itself easily due to the typical small size of test data sets and the powerfulness of the developers' machines. I encourage using libraries and algorithms that have at least some documentation about their efficiency and to search for information about algorithm efficiency.
- **Choosing the right data structures** – data structures strongly determine the operation of the application. Some are designed to

be efficient for large amounts of data, and others break down with even small datasets. In addition, the data structure must support the proper execution of the software. Even an efficient data structure will not help if the application has to constantly go back and forth through it searching for information.

- **Optimising in the right place** – typically, there are a few locations in an application where a large proportion of the application's execution time and energy is spent. These should be identified and optimising them is beneficial. On the other hand, optimisation should not be evenly distributed across the entire application, and premature optimisation is harmful – that is, the functionality of the application must be understood in depth before excessive optimisation.

- **Refactoring** – if the implementation of an application does not meet the requirements, it may have to be rewritten partially, or refactored. This is a labour-intensive task that also easily leads to errors, so the requirements or benefits must be sufficiently significant. In refactoring, I recommend considering all the advice in this paragraph and applying it when possible. Comprehensive test coverage or strict typing help to ensure that the refactored application works as intended.

- **Changing the runtime environment** – there can be significant efficiency differences between the runtime environments of interpreted and bytecode languages. Usually the differences are between runtime environments from different providers, but upgrading to a newer version can also provide additional performance. Follow the development of runtime environments and choose the one that is best suited to your application.

- **Changing the implementation language** – sometimes it is reasonable to change part of the implementation of an application to another language for efficiency reasons. The most typical examples are routines implemented in C that are called from applications written in a scripting language. The solution is not

straightforward and all languages cannot easily be combined with each other, but especially in situations requiring heavy calculation, this might be a good option. On the other hand, implementing the entire application in a different language usually does not make sense due to costs.

- **Taking into account background execution** – both desktop and browser applications are often in the background while the user focuses on other tasks on their device. Both operating systems and browsers automatically alter their behaviour for background execution by adjusting prioritisation to squeeze down the available processor cycles, and modify, for example, the frequency and timing of executing scheduled tasks. In addition, background execution can be taken into account in your own code, as long as the application gets a signal for switching to background and returning to the foreground. When running in the background, the user interface is not updated and timers are set longer, for example, information is retrieved from background systems with lower frequency than normally.

7.5 Use External Solutions

To improve efficiency, it is also possible to use various external services that act as intermediaries between the server and the user's device. These solutions typically are designed for content management environments, but of course they can be used creatively elsewhere. Other external solutions are also available and their advantage is optimisations for specific tasks on code and sometimes also on hardware level.

- **Using a Content Delivery Network** – CDN stores data near the user, typically they have servers in local operators' backbone networks. The use of CDN is reasonable especially when the service has many users who are served the same information or files, or when the service is globally distributed and latency needs to be reduced. CDN stores data on its own servers around the world, so the amount of data multiplies.

- **Cache and load balancing servers** – There are solutions for fast storing and retrieving of temporary or permanent data for various purposes. They can be used, for example, to share intermediate results requiring a lot of processing power to compute among several application servers. In such a case, the increasing complexity of the application and the challenges of keeping the cached data up to date should be considered.
- **External encryption solutions** – If the system is widely used, using an external SSL accelerator might be advantageous. This may require introduction of new devices, but dedicated hardware designed for encryption may be significantly more efficient. Similarly, it may be beneficial to use an existing VPN solution to secure communication between internal applications instead of using encrypted connections within the application.

7.6 Other Solutions

The world is moving towards greater volatility in energy costs due to the increasing use of renewable electricity production methods. This combined with the energy crisis that has hit Europe due to the war in Ukraine has made the timing of energy consumption more significant. Volatility is unlikely to disappear within the next few years because energy storage is still in its infancy.

As a rule of thumb, when energy is cheap, it is also renewable. If there is a need to perform computations that are not tied to a specific time, it is best to schedule them for periods of cheaper energy. This way, both the clean energy consumption is maximised and the energy price peaks are evened out, which affects, for example, people in the most vulnerable position.

There are already such solutions and, for example, Carbon Aware SDK[36] found on GitHub, can be used to enable your own applications to adapt to the changes in the price of electricity.

[36] github.com/Green-Software-Foundation/carbon-aware-sdk

The energy consumption of development servers and environments can also be examined critically. For example, should staging and test servers be shut down at nights and on weekends? Are their configurations appropriate or are they too powerful considering their usage and therefore consuming too much energy? Further, the staging and test environments can be virtualised and containerised to use devices as optimally as possible.

Summarised

1. Minimise data processing; reduce application data processing, storage, and transmission by optimising data models, static files, and user inputs.

2. Decrease data transmission by optimising transmission frequencies, compressing data, selecting energy-efficient protocols, and reducing HTTP headers and redirects.

3. Reduce the application's size by removing unnecessary code, minimising library usage, and eliminating unnecessary or rarely used functions.

4. Choose appropriate algorithms and data structures, optimise critical parts of the application, refactor if needed, and consider switching execution environments or programming languages.

5. Critically assess the energy consumption of development servers, optimise testing and production environments, and contemplate virtualisation and container technology for optimal resource utilisation.

8 Special Solutions

In addition to the basic application code, there are several energy-consuming solutions in use today, such as artificial intelligence or blockchain. These need to be addressed separately because their energy consumption differs from that of traditional application code. In the worst cases, the use of such solutions can skyrocket energy consumption. On the other hand, there are also undeniable benefits to their usage. The crucial point is to identify how and why energy is being consumed and whether it can be addressed.

In this chapter, a few commonly used solutions are introduced, whose energy footprint can be significant.

8.1 Artificial Intelligence

Artificial intelligence (AI) solutions came into the public consciousness in the year 2022, particularly with the likes of ChatGPT and DALL-E making AI testing and usage more accessible without requiring specialised programming expertise. Correspondingly, expectations for the business impacts of AI grew exponentially. For instance, Gartner predicts that by the year 2024, 40% of enterprise applications will incorporate generative AI capabilities. Similarly, they forecast that 15% of applications will be authored by AI by 2027.[37]

[37] Gartner Experts Answer the Top Generative AI Questions for Your Enterprise, www.gartner.com/en/topics/generative-ai

In this section, various forms of AI, machine learning, and related terms are grouped under the umbrella concept of artificial intelligence.

AI involves computationally intensive processes and its energy consumption can be substantial.

More has been written about the energy consumption of artificial intelligence in recent times, but much like the treatment of typical software energy consumption, concrete figures are scarce – mainly because the companies developing AI don't disclose them publicly.[38] AI applications are often run as cloud services, concealing real energy consumption within service billing.

For instance, it's estimated that around 10-15% of Google's electricity consumption is attributed to AI usage.[39] In 2021, Google's electricity consumption was 18.3 terawatt-hours, which would place AI consumption somewhere between 1.8 and 2.7 TWh. Even the range of 0.9 TWh is a significant figure, roughly equivalent to a month's production of the Finnish Olkiluoto nuclear power plant's Unit 3, the most powerful nuclear plant in Europe.[40] Furthermore, this single unit cannot cover Google's AI electricity consumption, as its annual output is approximately 12 TWh.

As these examples show, consumption figures are rough estimates and the ranges are wide. However, this should not deter the assessment and minimisation of actual consumption.

---

[38] As the AI industry booms, what toll will it take on the environment?, www.theguardian.com/technology/2023/jun/08/artificial-intelligence-industry-boom-environment-toll

[39] Artificial Intelligence Is Booming—So Is Its Carbon Footprint, www.bloomberg.com/news/articles/2023-03-09/how-much-energy-do-ai-and-chatgpt-use-no-one-knows-for-sure

[40] www.tvo.fi/en/index/production/plantunits/ol3.html

8.1.1 Energy Consumption

The energy consumption of artificial intelligence can be divided into three parts:

- **Compilation and structuring of training material.** AI cannot be taught by inputting just anything. The computer-era principle of "garbage in, garbage out" applies to AI training as well. Therefore, training material must be compiled, organised, and cleaned up. Since a vast amount of training material is needed, this task must also be automated, which consumes energy.

- **Teaching AI.** Neural network-based AI forms connections based on training material. The amount of training material depends on the AI's implementation technology, desired skill set, and precision level. Similarly, the format of training material affects energy consumption in teaching – for example, going through video or image-based training material requires more power per unit of information gained compared to processing text-based material.

  For example, the GPT-3 model, which has 175 billion parameters, is estimated by researchers to have consumed 1,287 MWh for training and produced 552 tons of carbon dioxide emissions.[41] In the same article, it is also noted that AI energy consumption can vary by factors of one hundred to even one thousand depending on the infrastructure used. Similarly, the size of training data or the generated AI model does not solely determine energy consumption, as different AI models learn with varying energy efficiencies.[42]

[41] Carbon Emissions and Large Neural Network Training, arxiv.org/abs/2104.10350

[42] A Computer Scientist Breaks Down Generative AI's Hefty Carbon Footprint, www.scientificamerican.com/article/a-computer-scientist-breaks-down-generative-ais-hefty-carbon-footprint/

Code Carbon has measured the energy consumption of popular models developed for natural language generation and computer vision learning.[43] The differences are significant. Unfortunately, standardised energy consumption data for learning is not available. Hopefully, this issue will be addressed in the near future.

- **Energy consumption during AI usage.** When AI is employed to perform tasks, such as interacting with humans or analysing X-ray images, the utilisation of neural networks consumes energy. A neural network operates similarly to any other software code, where data is transferred from memory to the process for processing and then back. The size and complexity of the network lead to a substantial amount of this processing and data transfer.

  For example, a single query with ChatGPT is estimated to consume energy ranging from 1.7 to 2.6 Wh.[44] Official information regarding daily query quantities is not available. Several sources have estimated the daily query count to be around ten million, which would result in a daily power consumption of 17 to 26 MWh.

  Additionally, the energy consumption during usage should include the loading of the AI model into the server's memory and other preparatory steps for deployment. For instance, the estimated size of ChatGPT's GPT-3 model is around 800 GB, so loading it is not insignificant.[45]

In summary, it can be concluded that AI solutions can easily consume a significant amount of energy, and this consumption often remains hidden within cloud services. There are substantial differences in energy consumption among different solutions. This, in turn, leads to an

[43] mlco2.github.io/codecarbon/model_examples.html

[44] ChatGPT's energy use per query, towardsdatascience.com/chatgpts-energy-use-per-query-9383b8654487

[45] en.wikipedia.org/wiki/GPT-3

increasing responsibility for application developers in terms of AI usage and solution selection.

8.1.2 Recommendations

The use of AI can significantly increase an application's energy consumption by several magnitudes, so it's important to be cautious when dealing with it. I recommend considering the following questions:

- **Is AI even necessary?** Many problems are currently trendy to solve using AI, even though they could be addressed with heuristic or statistical methods. However, this might not be as media-friendly and could, for instance, impact the valuation of growth companies.
- **How should AI usage be constrained?** If AI is indeed required for some reason, it's worth contemplating how its usage should be constrained. It's just a tool, not an end in itself. This involves striking a balance between business needs, computational accuracy or verifiability, and energy consumption.
- **Which AI model is suitable for the need? How is it procured?** Different AI models are suited for different needs, and the model should be chosen based on the specific requirement. Opt for a model that's already in use, and if it's a cloud-based solution, aim to use a geographical location where electricity is produced as cleanly as possible.
- **How is the model trained? Where is the training data sourced and how is it prepared?** The training data should align with business needs, account for biases, and the creation process should be as energy-efficient as possible. If the AI is for internal use or doesn't require human interaction, there may be fewer potential misuse concerns. In such cases, harmful content generated by AI might be viewed differently than in a fully open solution.
- **How is the model configured?** AI models contain numerous parameters that modify their behaviour, some of which

significantly impact energy consumption. Sometimes, the heaviest computations don't necessarily yield the best results. Finding the right balance requires experimentation.

Finally, I encourage everyone working with AI to share their own experiences regarding the use of solutions and energy consumption. There is currently a dire lack of information on these matters.

8.2 Blockchain and Cryptocurrencies

Blockchain is a technology that allows systems to generate and maintain a shared database in a decentralised manner.[46] A blockchain consists of a list of transactions that are distributed among participants, and its integrity can be verified through cryptographic hashes included in each block. This enables participants to trust each other without needing prior acquaintance.

New transactions can only be added to the end of the blockchain. In doing so, the current end of the list is recorded as a cryptographic hash along with a timestamp and the transaction details. The new transaction becomes the new head of the list. Transactions in the list are practically immutable, as altering them would require recalculating the hashes from the modification point to the end of the list.

Blockchain networks are typically managed by peer-to-peer software that employs a consensus algorithm to decide which transaction gets added next to the list and how it's validated.

In distributed systems, the blockchain is copied in its entirety to each separate system. These systems communicate with each other to add new transactions and use a consensus algorithm to agree on the next transaction to be added to the list.

The first blockchain solution was introduced in 2008 alongside the launch of Bitcoin. Bitcoin has been followed by a variety of blockchain-based

[46] en.wikipedia.org/wiki/Blockchain

currencies collectively referred to as cryptocurrencies due to the cryptographic hash computation involved in their creation.

In addition to the aforementioned cryptocurrencies, blockchains are used in smart contracts and non-fungible tokens (NFTs).[47]

8.2.1 Energy Consumption of Blockchains

Blockchain is inherently an energy-inefficient solution due to its decentralised nature and the redundant storage of data. Its energy consumption arises from the following actions:[48]

- **Execution of consensus algorithms** – some consensus algorithms are computationally intensive, and their energy consumption may increase with the number of devices connected to the network. For example, the proof-of-work algorithm used by Bitcoin is based on the amount of computational work and consumes a significant portion of the world's electricity.
- **Redundancy** – since all systems related to the blockchain contain the entire blockchain, the required storage space grows with each new transaction. As the chain is immutable, it cannot be shortened without compromising its integrity.

  Additionally, each system handling the chain independently performs the same computational operations with each transaction.
- **Data transmission** – systems dealing with the blockchain must constantly communicate with each other to reach consensus on the next transaction to be added. The growth in the number of systems also increases the inter-system data transmission, generally

[47] en.wikipedia.org/wiki/Non-fungible_token

[48] An analysis of energy consumption and carbon footprints of cryptocurrencies and possible solutions, www.sciencedirect.com/science/article/pii/S2352864822001390

reducing the efficiency of data transmission. The efficiency is also influenced by the topology of the network formed by the systems – a denser network is more energy-efficient. Bitcoin's data transmission illustrates the inefficiency: over 98% of the transmitted data is redundant. In other words, less than 2% of the transferred data is new.

Since blockchains will never achieve the same efficiency as centralised systems, largely due to the decentralised nature of the solution and cryptographic computations, it is important to carefully consider the use of blockchain technology.

If blockchain does not provide a genuine advantage over a centralised solution, its use cannot be recommended due to energy consumption. Additionally, it should be noted that blockchain-based solutions are more complex to design, implement, and maintain due to their decentralised nature. On the other hand, decentralisation provides the solution with better overall availability compared to a centralised system, which relies entirely on the availability of a central component.

8.2.2 Energy Consumption of Cryptocurrencies

While cryptocurrencies are not typically part of applications, their usage results in significant energy consumption, making them a key topic in the context of blockchain technology.

The energy consumption of cryptocurrencies is based on the blockchain consumption model discussed in the previous section. However, different cryptocurrencies have implemented varying technical solutions that have significant impacts on practical energy consumption.

There are thousands of different cryptocurrencies[49], but their market value drops rapidly after the top contenders. As of August 2023, the largest cryptocurrencies by market capitalisation were:[50]

1. Bitcoin 578 billion USD
2. Ethereum 223 billion
3. Tether 83 billion
4. BNB 38 billion
5. XRP 34 billion

The market capitalisation of the tenth cryptocurrency on the list was only $7 billion. The energy consumption of a cryptocurrency is influenced by the chosen technologies and the currency's value – the higher the value, the more people are interested in participating in cryptocurrency mining.[51]

The least efficient consensus algorithm is the proof-of-work used by Bitcoin. In this algorithm, systems perform a certain number of complex calculations to prove their participation in connecting the next transaction. This work has no other value and its results are not used for any other purpose. As the number of machines processing Bitcoin and, consequently, computational power increases, the amount of work is increased proportionally. This is done to ensure that no one – not even a state actor – can provide 51% of the network's computational power and thereby take control of consensus.

This directly affects the energy required by Bitcoin. Based on the August 2023 figures, its estimated annual energy consumption is about 107 TWh,

[49] originstamp.com/blog/how-many-cryptocurrencies-are-there

[50] www.bankrate.com/investing/types-of-cryptocurrency/

[51] Bitcoin boom: What rising prices mean for the network's energy consumption, www.cell.com/joule/fulltext/S2542-4351(21)00083-0

which is equivalent to the energy consumption of the Netherlands. A single transaction consumes about 635 kWh of energy. Compared to a credit card transaction, a Bitcoin transaction is approximately 430 thousand times less efficient.[52] Additionally, it's important to consider the environmental footprint of the hardware used to compute Bitcoin. This hardware consists of highly specialised computational machines that may not have reasonable opportunities for reuse.

The second-largest cryptocurrency, Ethereum, transitioned from the proof-of-work model to the proof-of-stake model in September 2022. In the proof-of-stake model, participation is based on the amount of currency that an individual system is willing to stake as collateral to secure a transaction. This change was truly transformative. The estimated annual consumption of the proof-of-work model was about 83 TWh just before the transition, and afterward, it was only 20 MWh. An individual Ethereum transaction consumes approximately 0.03 kWh, which is equivalent to about 40 credit card payments.[53]

Unfortunately, there is no equivalent data available for the energy consumption of other major cryptocurrencies. The fourth-largest cryptocurrency, BNB, uses an efficient proof-of-stake-authority model, and the network's validation is closed – there are 21 servers responsible for verifying new transactions.[54] In May 2022, the energy consumption for a single transaction was estimated to be 0.008 Wh, which is less than the energy consumption of a credit card payment.[55]

As evident from the above examples, the energy consumption of cryptocurrencies varies significantly from one currency to another.

[52] digiconomist.net/bitcoin-energy-consumption

[53] digiconomist.net/ethereum-energy-consumption

[54] www.adan.eu/en/publication/blockchain-protocols-and-their-energy-footprint/

[55] opentaps.org/2022/03/24/estimating-the-energy-impact-of-the-binance-smart-chain/

Unfortunately, the inefficiency in energy use is not reflected in the popularity of Bitcoin. We are wasting a significant amount of energy – both fossil and clean – on a currency largely used for speculation.

If you own Bitcoins, the most environmentally responsible action is to leave them permanently in a cryptocurrency wallet. Selling them creates a new energy-consuming transaction, and the buyer will likely put the coins back into circulation. Another question is whether individuals can genuinely afford to support climate efforts with their Bitcoin holdings, which can be substantial.

Furthermore, I encourage you to consider the rationale behind cryptocurrency investments. There are few practical use cases, such as using cryptocurrencies as a means of payment; it's mostly about speculation. If you do want to engage with cryptocurrencies, explore their energy consumption aspect as well.

8.3 Internet of Things

The Internet of Things (IoT) refers to systems that enable devices to be remotely monitored and controlled based on the data they automatically generate, using an internet connection.[56] It also encompasses concepts like the Industrial Internet and smart homes, which are applications of the Internet of Things.

The data produced by the Internet of Things is aimed to be intelligently processed into actionable structures, such as transforming vast amounts of real-time analytics into meaningful metrics and derived status updates.

The number of IoT devices connected to the internet was estimated to be 14.4 billion in the year, and it's projected to grow to 29.7 billion units by the end of 2027. The average estimated annual growth rate is 16%.[57]

[56] en.wikipedia.org/wiki/Internet_of_things

[57] State of IoT – Spring 2023, iot-analytics.com/product/state-of-iot-spring-2023/

Currently, machine-to-machine connections are estimated to account for about half of the connections of internet-connected devices.[58] Smart home devices constitute the largest segment in connections, while internet-connected vehicles are the fastest-growing segment.

In addition to this, we must also consider IoT devices in closed networks, such as sensors operating within factory networks. Altogether, the number of devices has grown so large that even small changes are significant in terms of energy consumption.

8.3.1 Energy Consumption

Typical IoT devices, such as various sensors, are designed to be energy-efficient. They can be powered by primary or secondary (rechargeable) batteries. For example, a sensor could be embedded in a concrete casting or enclosed within a wall to monitor humidity. In such cases, it's important to consider whether the device's battery needs to be replaced every year or every ten years.

On the other hand, if a sensor receives power from a measurable device, energy consumption in design is no longer as significant a factor. The value of the information obtained from the devices is so substantial that energy consumption is not practically significant, unless it leads to the aforementioned manual work.

The amount of energy consumed by IoT sensors can be adjusted using the following parameters:

- **Sampling frequency** – how often the device takes measurements. Does it meter instantaneous measurements or trends over time? Or does the device only monitor events that exceed a certain threshold?

[58] Cisco Annual Internet Report (2018–2023) White Paper, www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html

- **Data transmission channel** – devices can connect to the Internet in various ways, such as using WLAN or mobile networks. Additionally, home automation employs protocols like Zigbee and Z-Wave. As discussed earlier, there are significant differences in energy consumption among these networks. For instance, the mentioned home automation protocols are considerably more efficient than WLAN.

- **Data transmission frequency** – whether data is sent to the recipient with each sampling or if multiple samples are combined into one packet for transmission. This approach saves energy related to data transfer, such as opening and closing communication channels. However, it sacrifices real-time capabilities. The device may also become more complex, and storing samples might require additional storage space, increasing the emissions generated during manufacturing.

- **Sample size** – how much information is included in each sample. This significantly affects data transmission consumption between the device and the receiving system. It might be reasonable, for instance, to compress a sample before transmission if data transfer consumes more energy compared to processing.

These parameters are closely tied to the nature of the measured process or variable and the use case. It is of primary importance to choose the right system with its devices and sensors – only after that should one start considering energy consumption control.

The data obtained from sensors needs to be stored, interpreted, and analysed. It might require filtering, and eventually, conclusions can be drawn from the resulting information, either by machines or among humans. This process also consumes energy, and consumption increases proportionally with the amount of data being processed.

It's a good idea to consider data retention periods and, where possible, use various solutions that reduce the amount of data, such as time series databases that thin out data towards the past. These can be used to

perform analyses of the present while preserving historical averages, for example, to detect anomalies.

It's also important to remember that unused data is waste. If business operations do not require tracking a particular aspect at the device level, it's sensible to consider a different approach to solving the issue rather than storing sensor data year after year just to be on the safe side.

8.4 Data

All of the aforementioned solutions generate and consume vast amounts of data. In addition to these, data is produced through analytics, instrumentation of application operations, and tracking human activities. Moreover, people continuously generate data through photos, videos, and texts, often using various social media services. Similarly, the generation, usage, and transfer of this data create additional analytics data.

Data is collected from practically every electronic aspect of our surroundings. We talk about data streams flowing into data lakes. From there, data is pumped into processing to derive business-related insights. This arrangement sounds almost poetic, and when properly leveraged, it indeed offers a competitive edge to data-centric companies.

However, if a company doesn't handle its data correctly or collects entirely irrelevant data, waste is created. Data collection frequency might be too high, data is collected and stored in inefficient formats, or information is retained unnecessarily for too long.

Furthermore, applications produced by companies generate, transfer, and consume more data from year to year; in other words, the data intensity of applications increases. The accuracy of device sensors grows, leading them to produce more data at once—for instance, the file size of mobile phone images increases alongside the growth in camera sensors' megapixels.

User habits in applications also change, and an increasing portion of global communication relies on videos, which have significantly poorer

information density compared to images or text. In this context, information density refers to the amount of conveyed information relative to the amount of transferred data in bytes. Similarly, each user wants to consume data at their own pace, rendering broadcast transmissions ineffective. Instead, data is sent to each user individually in unicast. This is why streaming movies and TV shows over the internet is much less efficient than broadcasting them over a terrestrial network.

All of this leads to an increase in the amount of data and its transmission. And because data accumulates by nature – it's stored rather than deleted – its growth is continuous and regrettably rapid.

8.4.1 Global Scale

Global data volumes are enormous, and the annual growth is staggering. According to IDC's estimate, in 2018 there were 33 zettabytes of data, and this amount is projected to grow to 175 zettabytes by 2025.[59] A zettabyte (ZB) is 10^{21} bytes or one billion terabytes.[60]

In the same IDC report, it is projected that a significant portion of data storage will shift from endpoint devices to centralised data repositories, such as data centres and the cloud. However, data generation still primarily occurs on endpoint devices, even though the creation of data in the centre and edge – on the network or in distributed servers – is also increasing slightly. This, in turn, leads to growing data transfer requirements between endpoint devices and centralised solutions.

According to the report, consumer-generated data accounted for 47% of all data in 2017 and is projected to steadily decrease to only 36% by 2025. The remaining data is generated by businesses.

---

[59] The Digitization of the World From Edge to Core, www.seagate.com/files/www-content/our-story/trends/files/idc-seagate-dataage-whitepaper.pdf

[60] en.wikipedia.org/wiki/Byte

Unfortunately, there is no statistical information available on the energy consumption of data storage, and various estimates are outdated and include both storage and data transfer costs. Data transfer was discussed earlier in this book, and data transfer volumes will be addressed below.

In the evaluation of storage, calculations are based on formulas that use the average capacity of storage media – either hard drives or SSDs – and the energy consumed by the devices.[61] As of 2020, the consumption of hard drives is 0.65 watt-hours per terabyte-hour (storing a terabyte for an hour), and for SSDs, it's 1.2 Wh/Tth. This can be used to calculate an annual consumption of 5.7 kWh/Tta and 10.5 kWh/Tta.

IDC's estimate for the global total data volume for the writing year of this chapter – 2023 – is 100 zettabytes, which would mean that storing this data once would require 570 terawatt-hours (TWh) of energy per year, according to the aforementioned calculation.

Global annual electricity consumption is 25,530 TWh.[62] Therefore, the global electricity consumption of data would already account for 2.2% of the world's consumption. And this is for data stored only once and fully optimised, without any transfers or processing. Thus, it can be concluded that the calculation formula cannot hold true and significantly overestimates the actual energy consumption.

Additionally, data replication should be considered. Especially in cloud services and data centres, multiple copies or replicas of data are stored as a precaution to ensure data availability in the event of hardware failures.

---

[61] www.cloudcarbonfootprint.org/docs/methodology/

[62] World Energy & Climate Statistics – Yearbook 2023, yearbook.enerdata.net/electricity/electricity-domestic-consumption-data.html

The exact numbers of replicas used in cloud services are not clear, but according to one estimate, they range between one and six.[63]

Global monthly data transfer was estimated to be 180 exabytes in 2019 and increased to 230 exabytes in 2020, according to a UNCTAD report.[64] By 2026, data transfer is projected to be approximately three times that amount, reaching 780 exabytes. An exabyte is 10^{18} bytes, or one million terabytes. The annual data transfer in 2019 was 2.2 zettabytes, which seems quite small compared to IDC's estimate of 40 zettabytes for the total data volume in the same year.

In the same report, it is estimated that mobile data transfer will grow faster than fixed network data transfer due to the increase in the number of mobile phones and IoT devices. By 2026, mobile data transfer is expected to account for about one-third of all data transfer.

Consumer data transfer will continue to grow based on both increased device usage and the intensity of data usage. According to an Europe focused report by Arthur D. Little, the time spent online is expected to level off at around three to four hours per day in developed countries, but usage intensity will continue to grow for several reasons:[65]

- Improvement in video resolutions
- Wider use of videos on social media
- Expansion of internet usage into new scenarios – such as virtual meetings, attending concerts, or shopping

---

[63] Cloud Carbon Footprint Replication Factors, docs.google.com/spreadsheets/d/1D7mIGKkdO1djPoMVmlXRmzA7_4tTiGZLYdVbfe85xQM/edit#gid=735227650

[64] UNCTAD Digital Economy Report 2021: Cross-border data flows and development: For whom the data flow, unctad.org/system/files/official-document/der2021_en.pdf

[65] The Evolution of Data Growth in Europe, www.adlittle.com/en/insights/report/evolution-data-growth-europe

- Use of AR and VR solutions
- Growth in content generated by artificial intelligence

The biggest driver of growth is video consumption. According to the same report, video consumption accounted for 60% of mobile data transfer in 2022, and this share is projected to increase to 72% by 2030. In the same period, individual users' total mobile data transfer is estimated to grow from 15 gigabytes per month in 2022 to 75 gigabytes per month, with an annual growth rate of 25%. This means that in 2022, videos consumed an average of nine gigabytes of mobile bandwidth per month, and by 2030, the estimated data volume of videos is expected to be 54 gigabytes.

Similarly, through fixed connections an average of 225 gigabytes were transferred per month – per household, not per individual. It is estimated to increase to 900 gigabytes by the year 2030, an annual growth of 20%. Videos make up 67% and 74% of this data, which means that 151 GB of videos were transferred in 2022 and 666 GB in 2030.

While the energy efficiency of data transfer is expected to improve with the adoption of new technologies, it will not be possible to solely address the growth in the amount of transferred data without a revolutionary breakthrough. Hopefully, during this time, transitioning to renewable energy sources will significantly reduce the carbon dioxide emissions from the energy used by networks.

Changes in consumer behaviour and data usage are not the only drivers of data growth. Many rising solutions, such as artificial intelligence or online advertising, require significant amounts of data to function.

Training data for artificial intelligence is massive, and as the precision requirements of AI systems increase, dataset sizes continue to grow. Similarly, training data for AI needs to be constantly replenished, otherwise AI systems would gradually fall behind current events or wouldn't be able to generate images or videos following the latest artistic trends.

For instance, the size of the Internet archive and data collection produced by Common Crawl was 380 terabytes in October 2022.[66] This archive constitutes approximately 60% of the training data for the GPT-3 artificial intelligence model underlying ChatGPT.

Despite some partial discrepancies in the numbers, all these figures are significantly large, and each of them is growing substantially from year to year. Managing data volumes is one of the greatest challenges in sustainable IT, as data transfer, storage with replication, and processing contribute a significant portion of the energy consumption in the IT industry, and consequently, emissions as well.

Summarised

1. Apart from basic application code, energy-consuming solutions like artificial intelligence (AI) and blockchain must be addressed separately due to their varying energy consumption. While they offer benefits, their energy impact must be understood.

2. AI has gained prominence with applications like ChatGPT and DALL-E, but it consumes substantial energy. Gathering training material, teaching, and usaging AI all contribute to its energy consumption.

3. Blockchain's decentralised nature and redundancy make it energy-inefficient. Cryptocurrencies also consume significant energy, with different currencies using varying technologies leading to differing energy consumption.

[66] en.wikipedia.org/wiki/Common_Crawl

4 IoT devices generate data in a decentralised manner, and their energy consumption depends on several factors – such as sampling frequency and data volume, transmission method, and post-processing. The increasing number of IoT devices raises concerns about energy usage.

5 Data production, transfer, and consumption are rapidly increasing. Global data volumes are immense, with data replication adding to energy usage. Trends like video consumption and resolutions – along with new solutions like AI and blockchain – contribute to data growth and energy consumption.

9 Impact Assessment

The optimisation methods mentioned in the two previous chapters and adjustments for components consuming a lot of energy may help a little or a lot, depending on the nature and internal structure of the software. Some of them may be very easy and others extremely difficult to implement. This chapter discusses two different ways to assess changes and costs in order to make informed decisions about beginning a change project.

A monetary assessment is usually the most effective because with sufficiently large and credible numbers you can always get the attention of top management. Both CFOs and CEOs largely understand the world through numbers, and if you can get them on board with an idea, many obstacles have already been overcome. Unfortunately, there is no general formula for monetary changes, only the basics.

In addition, applications are implemented to cater for different needs. The optimisation of a solution used infrequently cannot be invested in to the same extent as an application that is delivered to the masses and used daily. The daily users or usage time of an application can be used as a multiplier to estimate the impact.

Since the execution time of software correlates strongly with the energy used – regardless of the implementation language – it can safely be said that improving the execution time reduces the energy consumption or costs of cloud services in the same proportion. The savings from such an operation can easily be converted into euros or dollars. Keep in mind that

the savings accumulate every year if the software usage remains the same. And if usage increases – as it usually does – the accumulated savings increase accordingly year by year.

Some savings come from postponing expensive changes, such as duplicating or load balancing the servers. When a service is able to serve more concurrent users, it may remain simpler for a longer time than a solution that can serve fewer users.

9.1 Impact vs. Workload

If it is difficult to quantify the effects of a change, it can also be approached using a 2x2 matrix approach, as shown in Figure 3 below. The time required for each proposed change is evaluated on a scale of small to large, and similarly, the energy savings or impact of the change is evaluated on the same scale.

Each organisation can decide for itself what a small or large workload or impact means. Businesses have different needs and circumstances, so no general metrics can be given.

For example, if a company operates in an energy-intensive industry such as metal processing or concrete production, one percent energy consumption decrease in the main process through software may be such a large reduction that the energy used by the application is of secondary importance. On the other hand, a decrease in energy consumption for a company that is entirely based on software may be huge for them, even if it may be way smaller than the reduction in the previous example.

The impact of the change must be genuinely substractional, that is, the same reduction cannot be achieved without the change, and the change does not happen in any case – otherwise the organisation would be shooting itself in the foot or engaging in greenwashing. The combined reductions of several potential changes may not necessarily be the sum of the individual reductions of each change. It is important to consider this when deciding on changes.

Since application developers are also only human, they may have different types of preoccupations and biases that either favour or oppose certain types of changes. Sometimes such biases are clear to the person themselves, but many of them are also unconscious. Thus it is a good practice to ask for at least two evaluations from different experts on the same change. If their thoughts are seriously mismatched, it will certainly lead to an interesting discussion that results in more refined thinking about the matter.

Below is a simple 2x2 matrix and various decision examples based on estimated workload and impact.

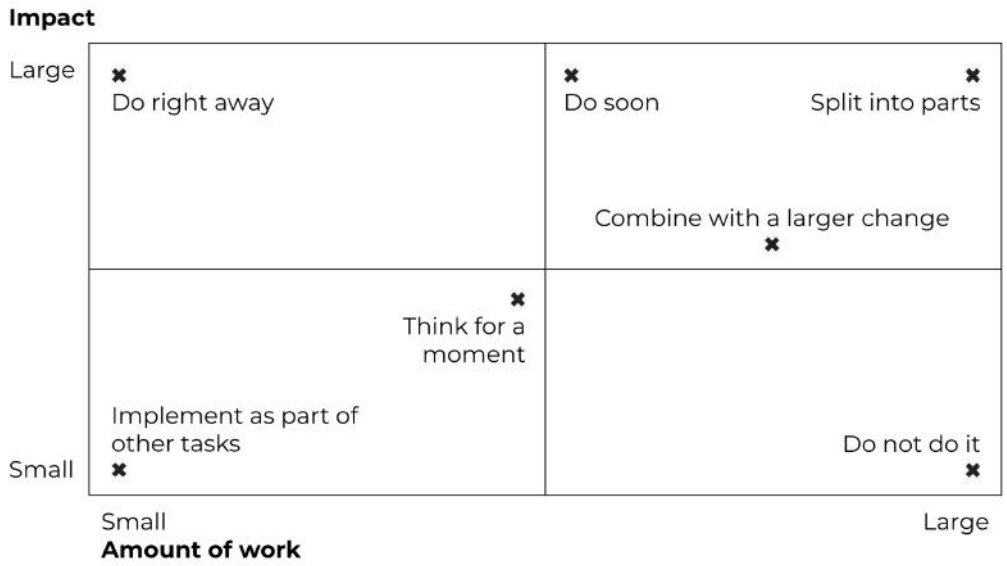


Figure 3. Assessing workload and impact on a 2x2 matrix.

9.2 Impact vs. User Experience

Simply evaluating the impact of a change in terms of the workload of application developers is not enough, because the change may have larger impacts beyond just the amount of work and cost; such as a change in user experience. If this is not the case and the change is purely technical, the following 2x2 matrix is not needed. If, on the other hand, the user

experience changes, it should be evaluated accordingly together with the expected impact.

In this 2x2 matrix, the user experience can degrade or improve. It is easy to assume that measures taken to reduce energy consumption will degrade the user experience – they are reductions, which, as a word, has a negative connotation. But if a change, for example, decreases the amount of information required from the user or otherwise simplifies the use of the service, the change can be significantly positive.

The 2x2 matrix should not be left to the developers to fill out, because they may not have the expertise or perspective to evaluate the change with regards to user experience. Instead, their evaluation of the change is often done by gut feeling or reflects the difficulty of changing the user interface.

Secondly, properly evaluating the change also provides good starting points for communicating the change to users. Users will react differently when they notice that the quality of videos has declined compared to also reading a message about the change and its impact on the environment.

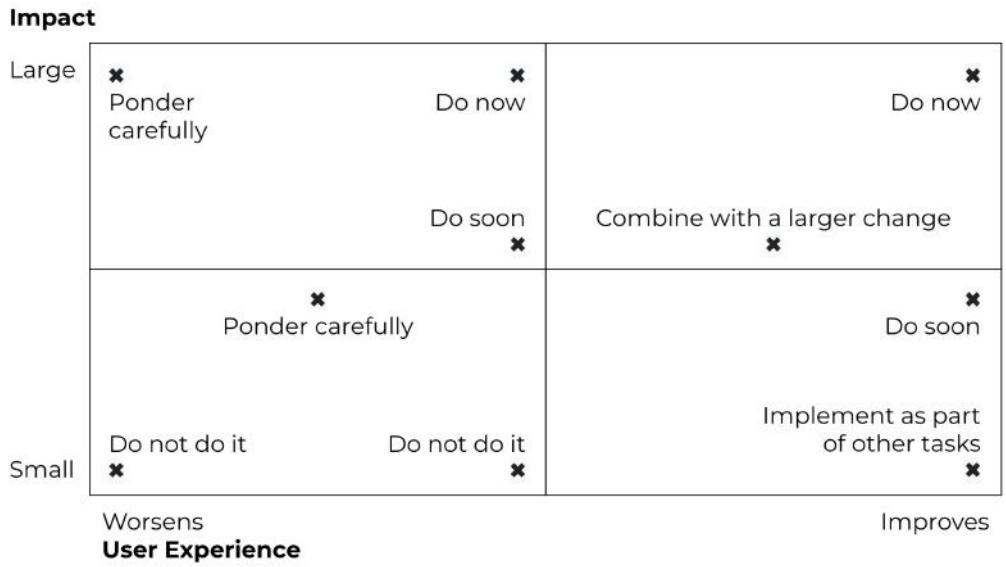


Figure 4. Assessing change of user experience and impact.

Summarised

1. Measure the changes with the credible numbers to capture the attention of the management, especially CFOs and CEOs who make decisions based on figures. Consider calculating annual savings as a basis for decisions.

2. Evaluate changes using an impact vs. workload matrix, based on workload and energy savings. Tailor the matrix definitions of "small" and "large" according to the organisation's needs.

3. Assess changes using a user experience change vs. workload matrix, taking into account improvements or deteriorations in user experience. Also, consider the need for communication and users' anticipated reactions in the evaluation of changes.

4. By reducing software execution time, energy consumption is decreased, and cloud service costs are likewise reduced; the savings can be easily quantified.

10 Carbon Neutrality

Carbon neutrality is a public goal for many organisations. It has an impact on the procurement of organisations, whose carbon footprint of purchased services is calculated as part of the buyer's footprint. As the carbon footprint will be included in IFRS reporting in the future for all three scopes, the carbon footprint of all companies following IFRS and their subcontractors must be calculated in a few years at the latest.

It is easiest if the organisation is carbon neutral and the footprint is zero. Carbon neutrality is typically achieved in three stages:

1. **Calculate the carbon footprint of your own operations.** It is generally calculated according to the GHG Protocol[67] and is divided into three scopes:

    a. Scope 1 – Emissions on site due to the company's own operations.

    b. Scope 2 – Indirect emissions related to purchased energy, such as emissions from electricity and heat production.

    c. Scope 3 – Emissions from the use of sold products, the procurement of goods and services, in other words indirect emissions. This category is divided into fifteen categories, from which the most important ones for the company's operations are chosen for the emissions calculation.

[67] Greenhouse Gas Protocol, ghgprotocol.org/

2. **Minimise the carbon footprint.** This is actually the most important stage of the process. Once the different components of the footprint are known, various actions can be targeted at them. For example, reducing fuel consumption by optimising logistics, cutting business travel by switching to video calls, or allowing continuous remote work. It is advisable to start with the biggest emission sources that can be affected. For example, changing the energy source for the office's air conditioning or heating may not be possible in rented offices, even if the energy consumption is significant.

   Minimisation should be done in several phases, as not all changes can be executed at the same time. Typically, the calculation is done annually, so the appropriate time frame for minimisation is also one year. The effects of the previous year's actions can be assessed and, based on the information, the next minimisations can be planned. Leading companies have been minimising for over a decade and still have opportunities left to reduce their own carbon footprint.

3. **Compensate for emissions that cannot be minimised.** There are several compensation methods available. The main idea is to remove carbon dioxide from the atmosphere to compensate for the amount emitted. As a result, the operation is theoretically carbon neutral.

   Compensation has been accused of being an indulgence and there are operators of varying quality in the market. For example in Finland, several parties have become aware of this problem and there will be more guidance and, at some point, requirements for the selection and implementation of compensation services. The advantage of compensation is that it sets a price for emissions, which is reflected in the company's profit and loss statement.

From the perspective of carbon footprint calculation, software falls under the third scope and the calculation of its carbon footprint as part of the company's carbon footprint is at least right now the company's own

decision. Some industrial companies, for example, only calculate scopes 1 and 2 in their own footprint. In the ICT industry, practically all emissions are in scope 3, unless the company in question is a data centre or teleoperator.

The carbon footprint of software can be divided into two parts: the implementation and maintenance of the software, and the carbon footprint of the energy and hardware used by the software.

10.1 The Footprint of Implementation and Maintenance

The carbon footprint of implementing an application is largely equivalent to that of normal office work. Software developers use computers in the same way as other office workers. In addition, the carbon footprint of systems supporting software development, such as version control, testing servers, ticketing, and the build pipeline, should be taken into account.

Projects may share resources, and the carbon footprint of these resources should be divided according to usage. This may not be easy, so various estimates are also acceptable. It is important to ensure that the carbon footprint is divided as a whole, but not multiple times (double counting).

As I previously stated, the discussion of the carbon footprint of the software developer versus the carbon footprint of the application is largely meaningless. Both should be reduced, and these reductions are seldom connected to each other. For example, a software developer riding a bicycle to work is unlikely to reduce their ability to write green code. On the contrary; when muscles do more work, the brain gets more oxygen.

10.2 The Application Footprint

The carbon footprint of an application consists of the energy used and the lifecycle emissions of the required hardware. Sometimes energy usage of a

device is included in its lifecycle, so it is important to be meticulous to avoid double counting.

The energy consumption of an application is largely correlated with the amount of time the application is spending, regardless of the implementation language. So, an application performing faster is also more energy efficient. Similarly, the amount of data transferred affects the carbon footprint according to the transmission path. This book is written to improve the efficiency of applications and to avoid endless recursion, so earlier topics will not be covered again in this chapter.

10.3 Code from Finland Carbon Neutrality Label

In early 2022, Code from Finland[68] launched the carbon neutrality label[69], which software companies can use to announce that they have zero carbon footprint. It should be noted that the symbol covers the carbon footprint of software implementation and maintenance. The carbon footprint of the produced software itself is taken into account only if it is part of the company's operations – for example, an application offered by a SaaS service provider is included in their carbon footprint. Similarly, applications produced by software consulting firms are included in their clients' carbon footprint.

One of the purposes of the symbol is to help the buyers of software to make more climate-friendly purchasing decisions. Code from Finland hopes that one day the symbol will be unnecessary, because all software development will be carbon neutral.

[68] codefromfinland.fi/

[69] codefromfinland.fi/en/symbols/carbon-neutrality-label

Summarised

1. Carbon neutrality is a goal for some organisations and its importance most probably grows in the future. Neutrality target impacts procurement and reporting, and it includes three areas: emissions from operations, purchased energy, and indirect emissions.

2. Carbon neutrality is achieved through calculation, reduction, and offsetting of emissions. Reduction focuses on key emission sources and is done annually. Compensation methods vary in quality, and careful consideration is needed when choosing them.

3. Software carbon footprint falls under Scope 3. The footprint covers both implementation and maintenance, as well as energy and hardware usage.

4. Code from Finland has introduced a carbon neutrality label for Finnish software companies. The label covers the carbon footprint of software implementation and maintenance, aiming to support climate-friendly software procurement.

11 Recommendations

This chapter presents brief recommendations for the main stakeholders involved in green software development. Each recommendation is not intended to be followed literally, but the situation should be assessed through one's own circumstances. There is no one-size-fits-all solution. The closest one fitting to everyone is to stop using proof of work cryptocurrencies, as they consume an enormous amount of energy in proportion to the benefit.

11.1 For Software Developers

The written source code and, as a result, the finished software is as energy efficient as its developers can and want to write efficient code. The system requirements and constraints create certain frames for energy consumption, within which the energy efficiency of the code can manoeuvre. In principle, there is no upper limit to inefficiency, so it is essential to develop energy-efficient solutions within such frames.

Developers can be motivated to be energy efficient through various external mechanisms, but internal motivation is essential to achieve permanent change.

Proceed as follows:

- Take the matter seriously and take action accordingly.

- Find out how your application consumes energy and identify the best locations to implement energy-saving changes.
- Minimise data transmission and storage needs.
- Consider the reasonable use of libraries. Refactor unnecessary code out whenever possible. Do not build unnecessarily complex structures if using a simpler one can reach the same goal.
- Consider the impacts of your changes and compare them to the estimated workload and potential changes in user experience. Consult other experts, such as product owners and UX designers, if necessary.
- Challenge product owners and designers when they propose new features. Discuss actively with designers about their plans for energy consumption. Have an open dialogue about objectives that degrade energy efficiency.
- If the implementation includes artificial intelligence or blockchain components, familiarise yourself with their energy consumption. Conduct your own measurements if information is not available. If you are purchasing these as services, request energy consumption information from the supplier and guidance on how to reduce the solution's energy usage.
- If you are a senior level developer, provide guidance to less experienced developers and help them understand problems and appropriate solutions.
- Understand that not everything can be achieved and sometimes inefficient code has to be written or left in place. There is no open check for all changes. Do not get upset about this, instead focus on the next changes.
- Do not confuse your own carbon footprint with the carbon footprint of the software you are implementing. They are two different matters and there are separate, independent solutions for both of them. Take everything possible into use.

- Extend the replacement cycle of your devices. Avoid replacing them unnecessarily or choosing new devices without considering the energy efficiency of their production, logistics, and usage. Repair devices when feasible.
- Participate in the development of industry-wide solutions and in the discussion of the energy efficiency of software. Share your own experiences and ideas.

11.1.1 For Component Developers

If an application developer creates a component or library to be embedded in other applications, the above recommendations apply directly. In addition, it's important to consider the energy efficiency of the component, especially if it's widely used.

For instance, SQLite[70] is installed on all Android and iOS devices, Apple's Mac computers, Windows 10 systems, and all widely used browsers. Estimates of the number of databases run into the trillions. Therefore, even small changes in energy consumption are significant as they multiply with enormous coefficients.

Of course, not all components are as popular, but that shouldn't hinder you from considering energy efficiency in your project.

Proceed as follows:

- Engage in active discussions about efficiency within the project.
- Ensure a coherent and project-wide approach to addressing energy consumption and efficiency.
- Build a measurement system and conduct performance related tests from one change to another. If the changes are substantial or performance consistently declines, address the issue.

[70] www.sqlite.org/mostdeployed.html

- Actively address changes in energy efficiency of the libraries used by the component.

11.2 For Designers

The software is not just code, but its efficiency is also determined in the design of the user-visible functionalities and their visual appearance, sometimes even more than in the source code. Designers must also share a similar responsibility for software energy efficiency as developers and architects.

Proceed as follows:

- Take the matter seriously and take action accordingly.
- Understand both the business needs and the user's wishes. Try to solve these potentially conflicting requirements efficiently.
- Minimise the user's chances of making mistakes. Design and write the user interface as clear and straightforward as possible. Take care of accessibility, as it reduces errors made by users with limited mobility or other challenges.
- Think about what the most minimal solution would be. Challenge the product owners and software developers when they propose new features. Have an open dialogue about objectives that degrade energy efficiency.
- Actively discuss the energy consumption of your designs with software developers. Grow your understanding of the application's functionality and figure out what is expensive and what is economical in terms of energy consumption.
- Consider whether functionality can be partially or fully replaced with instructions and images that guide the user. Design the content to be as energy efficient as possible, while still taking care of it staying informative and valuable to the user.

- Try to eliminate problems that require new code entirely rather than finding solutions to them. Not all problems can be eliminated, but some can certainly be prevented.
- Understand that the most energy-efficient solution is not always feasible due to business reasons or user needs.
- Extend the replacement cycle of your devices. Avoid replacing them unnecessarily or choosing new devices without considering the energy efficiency of their production, logistics, and usage. Repair devices when feasible.

In addition, the book Sustainable Web Design mentioned earlier offers many tips for designing more energy-efficient software.[71]

11.3 For Testers and Quality Assurance

After application developers, quality assurance and testers are the next individuals who witness the software in action. At this stage, the software hasn't yet reached end users, making it easier to implement changes. This is why quality assurance and testing play a crucial role in ensuring the software's energy efficiency.

Proceed as follows:

- Take the matter seriously and take action accordingly.
- Incorporate tests related to software energy efficiency. These tests can be included alongside stress tests and performance measurement tests.
- Monitor changes in software energy consumption from one version to another. Address sudden increases in consumption promptly in collaboration with application developers. Similarly, share information about long-term changes and engage in active discussions about energy consumption.

[71] https://abookapart.com/products/sustainable-web-design

- Strive to identify waste as presented in this book within the version being tested and report them to application developers or designers.
- Design and build energy-efficient testing environments. Ensure their proper use. Aim to use devices efficiently and maximise their lifespan.
- Turn off all environments when not in use.
- Schedule resource-intensive load tests during periods of cheap energy whenever possible.
- Consider appropriate testing frequencies for various automated tests. Break down the testing criteria into multiple levels of criticality and determine which level of tests should be conducted at each stage of the application development process.
- If you're a senior-level tester, take care of less experienced testers and help them understand how to promote energy efficiency through testing.

11.4 For Software Companies

Software companies have great power to influence the energy used by software, as the name of their industry implies, they are responsible for designing and developing the software. To make an actual impact, upper management must be aware of the matter and see the business opportunity in producing green code. Hopefully, this book has helped to convince you that eco-efficient coding is the future.

Proceed as follows:

- Take the matter seriously and take action now, not in the future.
- Train your staff to recognise the difference between efficient and inefficient coding. This book can help with that.

- Look at the entire architecture and identify the right areas for change – don't suboptimize.
- Ensure that your staff's hardware is efficient in terms of energy consumption. Also, extend the life cycle of the devices and ensure their proper recycling or reuse.
- Investigate the energy efficiency of the systems you produce and search for improvement opportunities.
- Aim for carbon neutrality or at least minimise emissions throughout your business.
- Evangelise the matter to your customers and partners. Demand carbon-wise solutions from your partners. Replace them, if your message does not seem to be getting through.

11.5 For Buyers

Purchasing environmentally friendly software will be increasingly important and sometimes even critical in the future. It is worth changing your own processes and thinking early on so that you are not caught with your pants down when regulation comes into force.

If environmental considerations were taken into account in the purchase of software – which is currently not done to a great extent – both the planet and the company CFO would benefit. A more efficient software consumes fewer resources, energy, servers, connections, and so forth, which is a direct saving for the buyer of the software.

It is important to remember that buyers have a lot of power – according to the golden rule: "he who has the gold, makes the rules."

Proceed as follows:

- Get carbon-wise digital partners – design, development, maintenance, connectivity, and so forth. Demand implementations that respect the environment from them and take this demand into

account in your budgeting, too. Help them move in the right direction.

- Investigate how your digital solutions impact the environment and adjust them as needed. Focus first on their carbon handprint and then on their carbon footprint. But look at both.
- Look at the entire value chain, don't sub-optimize. If necessary, take the whole business network into account.
- Don't greedily try to do everything at once. Make changes at a reasonable pace so that you will endure through the whole process.
- Beat the drum and evangelise the issue.

Additional tips for procurement organisations:

- Establish procurement criteria suitable for your organisation to evaluate sustainability in general and carbon neutrality and green coding in particular.
- Adjust the weights or scoring of new and existing criteria according to the values of your organisation.
- Share your experiences with other buyers.

11.6 For Users

Typically, the energy consumption of applications begins with the user. If the application has no users, it basically waits with low energy consumption. It is worth approaching personal application use with curiosity and thinking about whether I could do with less. Would I be even happier if my mobile phone was not constantly stuck to my hand?

Are some of my repetitive tasks joyless and reminiscent of addiction? Am I constantly spinning in social media, but not really getting anything out of it? Identifying and breaking these harmful habits both improves mental health and reduces the energy consumption of the applications.

And last but probably most importantly: reduce watching videos. About 80% of internet traffic is videos and if their information content were read as text, the energy reductions would be significant. Be wary of apps that push videos to you in an endless loop.

Proceed as follows:

- Identify the time you spend with applications. Various screen time counters are helpful here. Reduce application use if possible. The most energy-efficient application is an unused application. If you feel bad or inadequate from an application, such as social media, stop using it and delete the application. Don't return.
- Reduce watching videos. Think about whether you can read or tell the same thing in text instead of a video or picture. Don't fill all communication with memes and animated GIFs, but use them sparingly.
- Decrease the frequency of updating your devices and take good care of them. Buy devices that are promised software and security updates for a long time. Consider repair instead of replacement, especially replacing the battery can give several good years to a phone or laptop.
- Connect your devices to a fixed network if possible, especially mobile broadband should be changed to fixed. Prefer WiFi to mobile data transfer, and also prefer 5G to 4G.
- Adjust your device's display to turn off earlier and remove screensavers – a power-off or dark screen is more eco-friendly.
- Don't use bitcoin and similar energy-wasting solutions. Remember that not all digital solutions are automatically better than analog or previously used ones.
- Instead of using apps, use small idle moments for dreaming. Tolerate boredom.

- Bring your friends and acquaintances to participate in the change. But don't pressure or preach, but attract them through positivity.

Summarised

1. In the development of energy-efficient solutions, it is crucial to approach the challenges of energy consumption individually and avoid using a one-size-fits-all solution for everything.

2. Developers, designers, testers, software companies, and buyers each have their roles in creating energy-efficient software. Each group should take their responsibilities seriously, collaborate, and actively participate in ecological practices.

3. All parties should prefer energy-efficient devices and focus on extending their lifecycle. When using devices, practices that consume as little energy as possible should be followed.

4. Application developers should actively advocate for energy efficiency, develop industry-wide solutions, and generally adopt a positive mindset towards change.

12 Summary

The energy consumed by IT solutions has increased significantly in recent years and this has been recognised only recently. The carbon handprint of software has been generally very positive – in other words, they have succeeded in transitioning other processes to be more efficient and emit less greenhouse gases. In order to achieve global goals in combating climate change, it is necessary to consider the energy consumption of IT solutions, too.

Currently, IT industry trends unfortunately lead in the wrong direction in terms of energy efficiency and the industry is not regulated in terms of energy consumption. These are sure to change in the future. At the moment, the industry is gathering expertise in producing green code and there are various programs on the subject going on. There is still little progress, but the development is accelerating.

The energy consumption of modern applications can be roughly divided into three separate parts: the consumption of services operating in data centres and their internal network, the energy consumed in transmitting data from the data centre to the user's device, and the energy used to process and display the data on the device. Each application is individual and the details and the weights of the model vary from application to application.

At the moment, there is unfortunately very little measurement data available on the efficiency of software. It is not yet possible, for example, to compare the efficiency of your own application to similar applications

or to the industry average. There is also no equivalent standard for the energy efficiency of IT devices as there is for home appliances and televisions.

However, energy consumption can already be minimised without measurements. Optimising the functionality of applications, minimising data volumes and eliminating unnecessary items all lead to better energy efficiency.

The energy consumption of applications can be analysed using the concept of waste. Identify features or actions in the application that do not produce value but only consume energy. Removing these sources of waste will improve the performance of the application. There are several types of waste. The types identified in this book are not necessarily a complete list – instead their number will increase as understanding and experience in energy efficiency increases.

Removing waste is not just the task of application developers, although some waste is technical in nature. Typically, there are business reasons or user experience needs behind the waste. These matters cannot be evaluated solely from the perspective of energy consumption, but rather, measured decisions and compromises must be made.

This book presents a range of solutions for improving efficiency and eliminating waste. Once again, it must be stated that solutions must be applied in the context of the application and no single solution is sufficient. Because there are business reasons behind developing applications, not everything can be changed at once. Two different matrices are presented in this book for assessing impact.

Artificial intelligence, blockchain and cryptocurrencies, IoT devices, and data in general are increasing energy consumption and emissions in the IT industry. Overall, addressing energy usage across these emerging technologies is crucial to ensure sustainable IT practices and reduce environmental impacts.

In addition to the energy efficiency of the application, it is also worth calculating and minimising the carbon footprint of software development.

To this end, Code from Finland has published a carbon-neutrality symbol. By following the criteria for the symbol, you will get good guidelines for handling your own carbon footprint, even if you are not applying for the symbol.

Recommendations have been listed for various stakeholders – software developers, designers, users, software companies, customer companies and their professional buyers. It is essential in all of these recommendations that action must be taken immediately.

13 Thank Yous

The idea of writing this book has been on my mind for a while, first as quite shapeless and finally as more analytical thoughts. Eventually, the book had to be written because the idea no longer gave me peace.

I became interested in green coding based on discussions held in the board of Code from Finland. Together with a member of the board, the CEO of Hiottu Ltd. Satu Lapinlampi, we began to investigate the situation. Satu's cheerful yet very goal-oriented style of advancing environmental issues has been energising to watch. Without her, I would not be on this journey.

The first excellent summary for us was provided by Lotta Toivonen, an expert at Sitra – the Finnish Innovation Fund, and I have exchanged thoughts several times with Lotta during the process. Another significant step forward was a seminar organised by Tieke – Finnish Information Society Development Centre, where I was able to speak about the matter for the first time and get to know other people who were having similar thoughts. Huge thanks to Tieke's Executive Director, Hanna Niemi-Hugaerts, for this.

Through these connections, I got to know Professor Jukka Manner of Aalto University, whose thoughts about simpler and therefore more efficient software resonated very strongly with me. Similarly, Professor Jari Porras of LUT University has been an extremely important discussion partner on these matters.

I finally was asked to join the steering group of Tieke's Green ICT Project and ended up being its chairman. Fortunately, I got to know Project Manager Antti Sipilä, who has been a valuable help in many discussions and linking to experts in the field.

Jusu Toivonen, who is leading the sustainability efforts at ski resort Rukakeskus Ltd., has served as an inspiration in how environmental matters can be significantly improved with determination, long-term plans and persistence. The conversations with him have given me confidence that I am on the right path.

I discussed various ideas and developed my own expertise in the field of responsibility with Anna Savisaari, a responsibility lead at Exove. Without these discussions, I would never have come to write this book.

Conversations with Exove's Sales and Marketing Director, Päivikki Kyykkä, led me to the idea of waste. This concept allowed me to jump over the last hurdle – I had long been wondering how to move the issue forward analytically, but without measurement data. The concept of waste, borrowed from Lean, provided a liberating approach to advance the eco-efficiency of code without continuous measurement.

Special thanks to the pre-readers of this book, from whom I received a significant number of new thoughts and comments to add to the text. The pre-readers were Valohai Ltd. CTO Aarni Koskela, Exove Ltd. CTO Kalle Varisvirta, Rebase Ltd. Senior Software Engineer Tommi Sinivuo, and Exove Design Ltd. Designer Katri Pakula. Exove's Growth Marketer, Essi Rostedt, proofread and corrected the Finnish text, and Data Consultant Vivi Mattsson, Developers Ilse Tervonen and Koray Dündar, and Principal Tech Lead Rihards Steinbergs proofread and corrected the English text.

The second edition pre-readers were Kalle Varisvirta and TietoEvry's Senior Software Architect Tommi Lehtinen. Exove's Growth Marketer Jessica Holmberg proofread and corrected the Finnish text, and Developer Panajis Rantala from Exove proofread the English version.

Without you, this book would be significantly shorter and of lower quality in general.

It goes without saying that I am responsible for all possible errors.

14 Feedback

I am pleased to receive feedback and new ideas for producing green code. The matter is important to both the development of the IT industry and decelerating climate change. Further, the topic is very important to me and thus I strive to improve the quality and meaningfulness of this book.

I want to know if you felt that a relevant topic was not discussed or it was addressed only superficially. I also want to learn more about new types of waste or tips to improve software efficiency. It goes without saying that all errors should be reported.

Based on the feedback provided for the first edition, testing and artificial intelligence have been written about. In addition, a few typographical errors you noticed have been corrected. Thank you to everyone who provided feedback!

If you want to improve the quality of my book or just write a positive comment about it, please fill the short questionnaire found behind the following link:

bit.ly/greencodefeedback

Thank you.

Exove and UpCloud Bring Carbon Neutral Web Services to the Market

Exove and UpCloud have decided to join forces to provide responsibly implemented cloud-based web services for locally or internationally operating companies. The partnership offers customers a reliable and modern way to manage content and web services cost-effectively and with high quality while protecting the environment.

The partnership takes into account not only customer needs but also the environment and society. In terms of responsibility, the focus is particularly on minimising the carbon footprint and environmental impact of web services. Together, we have developed a CO2 calculator that allows for monitoring of web and cloud services' emissions almost in real-time. The most significant aspect is the ability to measure and reduce the carbon footprint of the web service across the entire chain.

UpCloud's cloud services are available in 12 data centres worldwide, two of which are located in Finland. The company is known on the global cloud market for its high availability and performance. UpCloud offers customers a 99.99% service level agreement and the best performance and 24/7 support in the market.

At UpCloud, we have placed sustainable development at the core of our operations and as part of this, we also strive to neutralise our carbon footprint in the long term.

Exove has taken all of this even further with developing of web services and we are excited about this evolving partnership.

Antti Vilpponen,
UpCloud

If you are interested, please contact to *business@exove.com*

Do You Want to Change Your Organisation's Code to Be Green?

Exove provides training and coaching for software developers and the rest of organisation, as well as consulting for upper management in implementing energy-efficient and carbon-neutral IT systems. We audit systems and analyse a company's overall ICT architecture.

Training

We train application developers, designers, and architects. The training covers the basics of green coding, identifying waste, and rational optimisation of systems.

Consulting

We consult with company management on specific issues related to green code, assist in developing energy efficiency, and improve organisation's capability regarding green IT.

Architecture Analysis

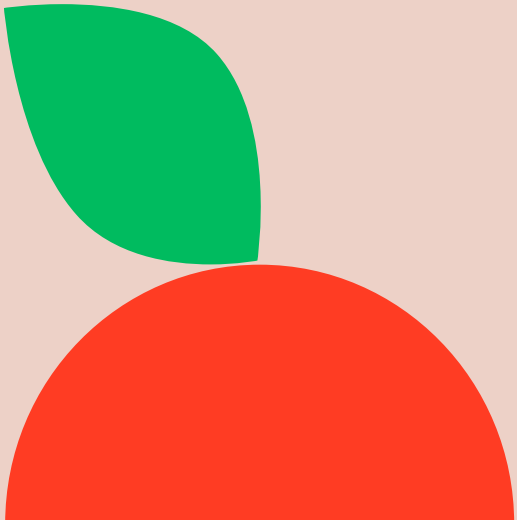
We analyse the overall ICT architecture from an energy efficiency perspective and create a roadmap for improvements. We also train ICT service providers as needed.

Auditing

We audit a company's current systems for energy efficiency and provide suggestions for improvements and an estimation of savings.

Coaching

We help bring the change of green coding to your organisation by coaching and guiding key players.



Interested? Get in touch:
business@exove.com