Tampere University

Vilma Salmikuukka

GUIDELINES FOR SUSTAINABLE SOFTWARE

Master of Science thesis in technology
Information Technology and Communication Science
Prof. Hannu-Matti Järvinen
CST Lead Satu Kaivonen
January 2024

ABSTRACT

Vilma Salmikuukka: Guidelines for sustainable software
Master of Science thesis in technology
Tampere University
Master's Programme in Information Technology
January 2024

Software applications contribute to greenhouse gas emissions indirectly through energy consumption, hardware production, and disposal of electronic waste. This work explores different factors that impact the environmental sustainability of software applications and investigates how these factors can be optimized during the software development process. The guidelines presented in this work include several strategies and actionable practices that can be incorporated throughout the software lifecycle.

To reduce the environmental impact associated with hardware, it is important to optimize the software for efficient utilization of hardware devices. The results of this work underscore the importance of developing software that leverages containerization techniques and cloud computing paradigms, ensuring optimal resource utilization across diverse computing environments.

This work also explores how software applications consume energy and investigates strategies to optimize energy usage from different perspectives. The findings emphasize the importance of minimizing data transfer and reducing the computational load within the software. By limiting the frequency and volume of data exchanges and reducing unnecessary processing, significant reductions in energy consumption can be achieved.

Various higher-level aspects of software development process are also covered in this work, ranging from project management and DevOps to user experience and interface design. Several optimization strategies are introduced in each area. The text also investigates how energy consumption is affected by different software implementation elements, such as architecture, programming languages, frameworks, data structures, and algorithms. Furthermore, the work provides a more focused examination of web development, offering detailed insights into front-end, back-end, and mobile development.

This work also highlights that the nature of the software influences the specific areas where optimizations can be applied, and which practices are realistic to implement. Therefore, the strategies and practices for sustainability need to be tailored to each application's unique characteristics and requirements.

Keywords: Environmental sustainability, Software development, Green Coding, Web applications

TIIVISTELMÄ

Vilma Salmikuukka: Guidelines for sustainable software
Tietotekniikan diplomi-insinööri
Tampereen yliopisto
Informaatioteknologian ja viestinnän tiedekunta
Tammikuu 2024

Verkkosovellukset vaikuttavat kasvihuonepäästöihin epäsuorasti energiankulutuksen, laitteiston tuotannon ja elektroniikkajätteen hävittämisen kautta. Tässä työssä tutkitaan erilaisia tekijöitä, jotka vaikuttavat sovellusten ympäristöystävällisyyteen ja selvitetään, miten näitä tekijöitä voidaan optimoida ohjelmistokehitysprosessin aikana. Työssä esitetyt suositukset sisältävät useita strategioita ja käytäntöjä, jotka voidaan ottaa käyttöön koko ohjelmistokehitysprosessin ajan.

Laitteistoon liittyvien ympäristövaikutusten vähentämiseksi on tärkeää optimoida ohjelmisto hyödyntämään laitteita mahdollisimman tehokkaasti. Työn tuloksissa korostetaan kontti- ja pilviteknologioiden hyödyntämistä, mikä auttaa varmistamaan optimaalisen resurssien käytön useissa laskentaympäristöissä.

Lisäksi työssä tutkitaan, miten sovellukset kuluttavat energiaa, ja selvitetään strategioita energian käytön optimoimiseksi eri näkökulmista. Työn tuloksissa painotetaan tiedonsiirron minimoimisen ja laskennallisen kuorman vähentämisen tärkeyttä. Rajoittamalla tiedonsiirron tiheyttä ja määrää sekä vähentämällä tarpeetonta prosessointia voidaan saavuttaa merkittäviä vähennyksiä energiankulutuksessa.

Työssä käsitellään myös ohjelmistokehitysprosessin korkeamman tason näkökohtia, kuten projektinhallintaa, DevOpsia, sekä käyttökokemus- ja käyttöliittymäsuunnittelua. Useita optimointistrategioita esitellään jokaisella osa-alueella. Tekstissä tutkitaan myös, miten erilaiset ohjelmistojen toteutuselementit, kuten arkkitehtuuri, ohjelmointikielet, kirjastot, tietorakenteet ja algoritmit vaikuttavat energiankulutukseen. Lisäksi työssä tarkastellaan tarkemmin verkkokehitystä ja käydään läpi yksityiskohtaisia näkemyksiä käyttöliittymä-, taustapalvelu- ja mobiilikehityksestä.

Työssä korostetaan myös, että ohjelmiston luonne vaikuttaa siihen, millä aihealueilla optimointeja voidaan soveltaa ja mitkä käytännöt ovat realistisia toteuttaa. Tästä syystä kestävyyden strategiat ja käytännöt on räätälöitävä kunkin sovelluksen ainutlaatuisten ominaisuuksien ja vaatimusten mukaan.

Avainsanat: Ympäristöystävällisyys, Ohjelmistokehitys, Vihreä koodi, Verkkosovellukset

PREFACE

This thesis marks the end of my studies at Tampere University. It was a long journey, but thanks to an amazing group of friends, it was filled with laughter, love, and unforgettable moments. I am also grateful to my family for their encouragement and support throughout my studies. Additionally, a special thanks goes to my boyfriend for always helping me and teaching me new things and being so patient with me. I could not have done this without him.

Furthermore, I want to thank CGI and Satu Kaivonen for making this thesis possible and providing such an interesting topic.

Tampere, 29 January 2024

Vilma Salmikuukka

CONTENTS

1. INTRODUCTION 1
2. ENERGY 2
 2.1 Carbon Intensity 2
 2.2 Software energy consumption 4
 2.3 Measuring energy consumption & carbon emissions 5
  2.3.1 Metrics 5
  2.3.2 Tools 5
3. HARDWARE 8
 3.1 Embodied carbon 8
 3.2 Extending lifespan 9
 3.3 Static power draw & Energy proportionality 10
 3.4 Increasing utilization 11
  3.4.1 Virtualization 12
  3.4.2 Containerisation 13
4. DEVOPS & PROJECT MANAGEMENT 15
 4.1 Cloud 15
  4.1.1 Power Usage Effectiveness (PUE) 15
  4.1.2 Data center location 16
 4.2 CI/CD 18
 4.3 Methodologies 18
5. WEB APPLICATION DEVELOPMENT 21
 5.1 Front-End Development 21
  5.1.1 DOM 22
  5.1.2 Frameworks 22
  5.1.3 WebAssembly 23
 5.2 Back-End Development 23
  5.2.1 HTTP 23
  5.2.2 Database 24
  5.2.3 Frameworks 24
 5.3 Mobile Development 25
6. SOFTWARE IMPLEMENTATION 28
 6.1 Architecture 28
 6.2 Programming language 30
 6.3 Concurrency 34

6.4 Data Structures & Algorithms 34
6.5 Cache 37
6.6 Synchronization 38
6.7 Data format 38
6.7.1 Text-based data 38
6.7.2 Images 39
6.8 Code implementation 40
7. USER EXPERIENCE 42
7.1 Usability 42
7.1.1 Content 42
7.1.2 User Preferences 43
7.2 User Interface Design 43
7.2.1 Images 44
7.2.2 Videos 44
7.2.3 Colors 45
7.2.4 Fonts 46
8. CONCLUSION 47

LIST OF SYMBOLS AND ABBREVIATIONS

ICT	Information and Communications Technology
GHG	Greenhouse gas
CO_2	Carbon dioxide
kWh	Kilowatt-hour
Elec.	Electricity
ADP	Abiotic resource depletion
MIPS/W	Million instructions per second per watt
FLOPS/W	Floating point operations per second per watt
HTTP	Hyper Text Transfer Protocol
GB	Gigabyte
SCI	Software Carbon Intensity
OS	Operating System
DevOps	Development and Operations
CI/CD	Continuous Integration and Continuous Deployment
PUE	Power Usage Effectiveness
SPA	Single-Page Application
SSR	Server Side Rendering
SSG	Static Site Generation
DOM	Document Object Model
Wasm	WebAssembly
CPU	Central processing unit
PWA	Progressive Web Application
OOP	Object-Oriented Programming
API	Application Programming Interface
UX	User Experience
UI	User Interface
IA	Information Architecture
OLED	Organic Light-Emitting Diode
LCD	Liquid Crystal Display

1. INTRODUCTION

The purpose of this work is to find out what are the current recommendations and guidelines for sustainable software. The research objectives for this work are to explore the factors that impact the environmental sustainability of software and to determine how these factors can be optimized and integrated into the software development process. These research goals are achieved by conducting a comprehensive review of the current literature on the subject.

Software development is one of the only engineering fields where environmental sustainability is only rarely taken into consideration. It is a common misbelief that digital solutions and technology are inherently clean and environmentally sustainable. This is understandable since software itself is not a source of pollution, as it doesn't produce physical waste or emissions. However, as software requires energy to operate and hardware to operate on, it contributes to pollution indirectly through energy consumption, electronic waste, and the production and disposal of hardware.

The Information and Communications Technology (ICT) sector has been estimated to produce 4 % of the total greenhouse gas emissions (GHG) and consume 10 % of the electricity of the world. This includes the manufacture and power usage of 34 billion digital devices, and the millions of kilometers of cable they are connected with [1] [2]. Although the ultimate responsibility of hardware and energy consumption is always with the device, software dictates the way hardware is utilized and energy is consumed [3].

Recently government policies and institutional standards have begun moving towards sustainable software. This work aims to gather guidelines for every state of the software lifecycle with actionable patterns and tools for decarbonizing software. While the guidelines have been generalized as much as possible, the main focus of this work is web application development.

In the context of this work, sustainable software means ways to make software itself greener, as opposed to software that encourages environmentally sustainable movements. However, it is worth noting that software also has the potential to significantly reduce GHG emissions through innovative solutions for digitalization, such as online meetings which reduce the need for traveling [4].

2. ENERGY

Green transition and the digitalization that follows require a lot of energy. This chapter briefly explains how energy consumption contributes to carbon emissions through carbon intensity, and how emissions and energy consumption are distributed between different devices. Following that, different metrics and methods are introduced for measuring emissions related to software.

2.1 Carbon Intensity

Carbon intensity measures the amount of carbon dioxide (CO_2) emitted per kilowatt-hour (kWh) of electricity produced. It is used as a key metric in assessing the environmental impact of energy production. Carbon intensity can be influenced by the source of the electricity, the efficiency of the energy conversion process, and the emissions associated with extracting and transporting the electricity source. [5]

Electricity that is generated using fossil fuels (coal, oil, and natural gas) is more carbon intensive while nuclear power and renewable energy sources, such as hydropower, wind, and solar, are defined as low-carbon energy sources. The percentage of electricity generated from burning fossil fuels varies by country and region, as visualized in Figure *1*. Globally, fossil fuels have traditionally been the dominant source of electricity generation. [6]

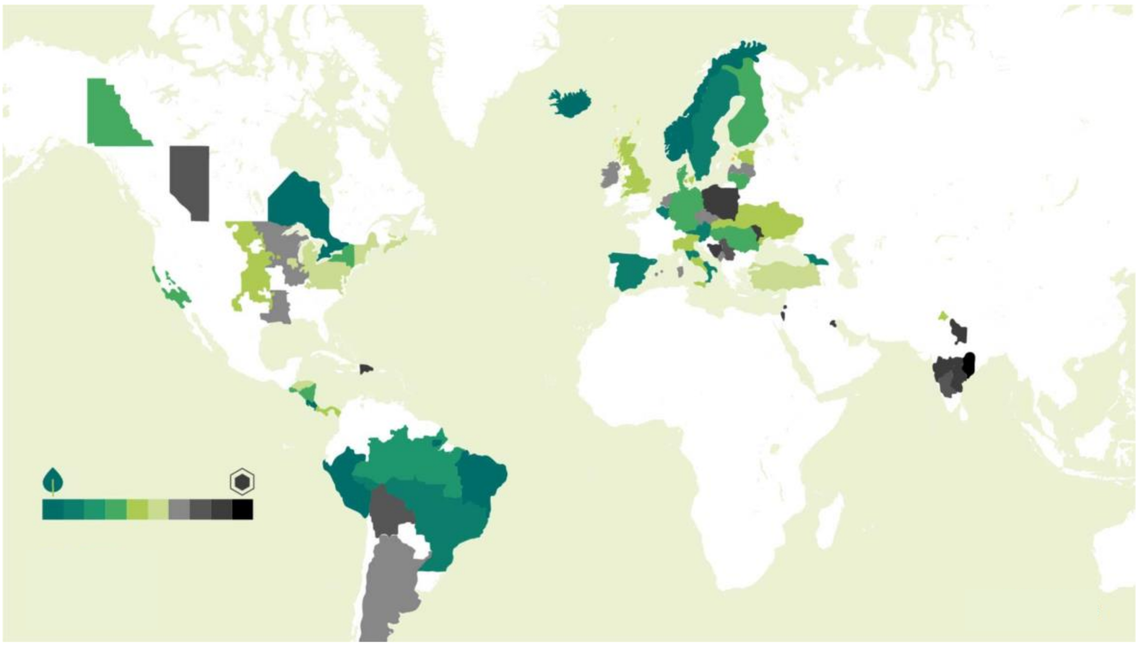


Figure 1. Visualization of the variability in carbon intensity in different regions [6].

Fossil fuels account for about 80% of global electricity production. Oil is the largest source of electricity from fossil fuels, followed by coal and natural gas. The remaining electricity comes from renewable energy sources and nuclear power. [7] [8]

Carbon intensity also changes over time due to the inherent variability of renewable energy caused by the changes in weather conditions as illustrated in Figure *2* [6].

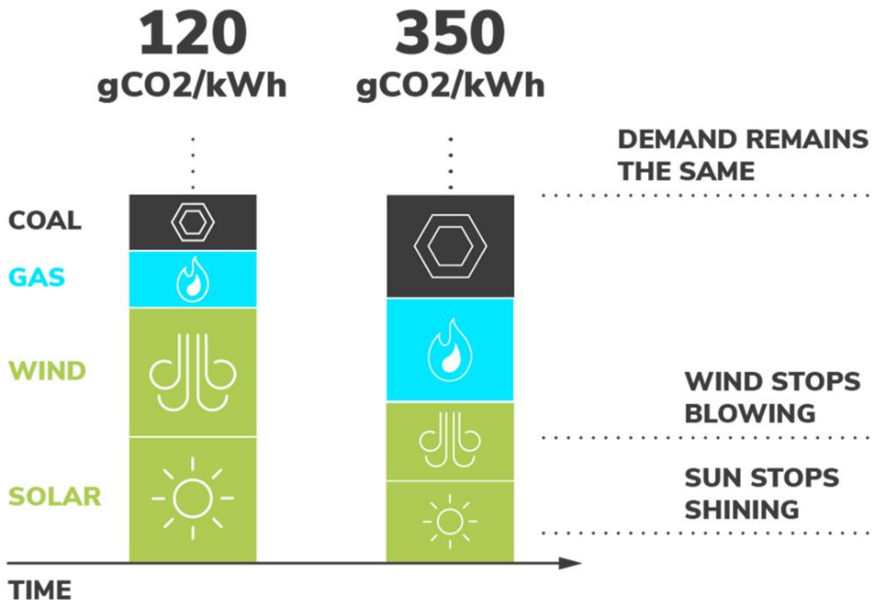


Figure 2. Visualization of the variability in carbon intensity due to weather conditions [6].

Unfavorable weather conditions for renewable energy sources can lead to a decrease in their contribution to the energy mix. In such cases, the energy demand is often met by conventional power plants, which typically have a higher carbon intensity. Shifting computation to a time and a place where weather conditions are favorable and renewable energy is available, can help decrease emissions [9].

2.2 Software energy consumption

Data centers, telecommunications equipment, and user end devices all consume energy as a result of running software. These devices contribute to other forms of pollution as well, such as water usage and mineral mining. Table *1* presents how different forms of pollution are distributed between the devices. The emissions related to the manufacturing process are also accounted for.

Table 1. Emissions distribution [1].

	Energy	GHG	Water	Elec.	ADP
User equipment	60%	63%	83%	44%	75%
Network	23%	22%	9%	32%	16%
Data centres	17%	15%	7%	24%	8%

As Table *1* shows, devices located in data centers consume the least amount of energy, only about one-sixth of the total amount. Network devices are not far behind, consuming nearly a quarter of the total amount. However, users' devices dominate energy consumption, by consuming over half of the total energy. Producing environmentally friendly software can influence the energy consumption of all this equipment.

2.3 Measuring energy consumption & carbon emissions

The environmental footprint of a software can be measured through different characteristics of sustainability, such as energy efficiency, hardware efficiency, and resource optimization [10]. Continuously measuring the emissions throughout software lifecycle, from design and coding to deployment and maintenance, can help identify critical components and monitor progress towards improvement. The obtained results can be compared to previous measurements or those of competitors [11].

2.3.1 Metrics

In most major industries, the key metrics of environmental performance and energy efficiency have been well-established and widely adopted. These can be, for example, liters per 100 km for cars or energy per square meter for homes. The tools and techniques used for calculating these metrics have become standardized, ensuring consistency and uniformity among environmental evaluations [12].

Various environmental standards and efficiency metrics exist for hardware, such as MIPS/W (million instructions per second per watt), and FLOPS/W (floating point operations per second per watt). However, no equivalent standards have been established for software [13]. Defining clear metrics allows for a better understanding and comparison of the measurements.

In general, energy efficiency can be defined as **useful work done per energy consumed** [14]. As the *useful work done* varies greatly depending on the software and even within different parts of the software, the unit of efficiency has a wide range of possibilities [15]. These can be for example, the number of sorted items per Joule, HTTP requests per second, or Joules per database query [16].

Different factors need to be considered when measuring the overall sustainability and carbon emissions of the software. Measures, such as the amount of data transferred (kWh/GB) and the carbon intensity of energy (CO_2/kWh), can be important indicators of carbon emissions. [12]

2.3.2 Tools

In order to determine the total carbon emissions of a software application, it is essential to have detailed information on energy consumption, carbon intensity, and the specific hardware utilized by the software. This information can be difficult to acquire [17]. However, there are several tools that can give indicative information about the environmental footprint of the software. Some of these tools are listed in Table 2.

Table 2. Tools for assessing the environmental footprint of software.

Tool	Description	Type	Ref.
Website Carbon Calculator, Ecograder	Estimate carbon footprint of websites.	Free, online	[18] [19]
Greenspector	Measure and analyze the environmental impact of mobile and web applications.	Commercial	[20]
PowerAPI	Measure power consumption of applications.	Free, open-source	[21]
CAST Highlight	Analyze application and source code for energy consumption patterns and make suggestions for improvement.	Commercial	[22]
Lighthouse	Google chrome developer tool for analyzing the performance of a web page.	Free, open-source	[23]
Emissions Impact Dashboard	Monitor the carbon impact of cloud in Microsoft Azure.	License	[24]
Carbon Footprint	Measure, report, and reduce cloud carbon emissions in Google Cloud.	Commercial	[25]
Cloud Carbon Footprint	Cloud carbon emissions measurement and analysis tool supporting multiple cloud providers.	Free, open-source	[26]
Software Carbon Intensity (SCI) Specification	A specification that describes how to calculate a carbon intensity score for software applications.	Free, open-source	[27]

Different tools can give insights on different aspects of the environmental footprint of the software. Using several different tools in a versatile way and combining the results can give a better overall understanding of the total emissions.

3. HARDWARE

One aspect of software sustainability is designing software to be hardware efficient. This chapter provides an overview on how hardware relates to carbon emissions and consumes energy. Different options are introduced on how software can utilize hardware more efficiently.

3.1 Embodied carbon

Hardware contributes to pollution through usage, manufacturing, and disposal of devices. The production of hardware often involves the extraction of natural resources and the use of toxic chemicals. The disposal of hardware often produces electronic waste as devices may contain hazardous materials [28]. Embodied carbon measures the amount of pollution emitted during the manufacturing and disposal of a device. Total carbon pollution means the embodied carbon and the pollution associated with using the device [29].

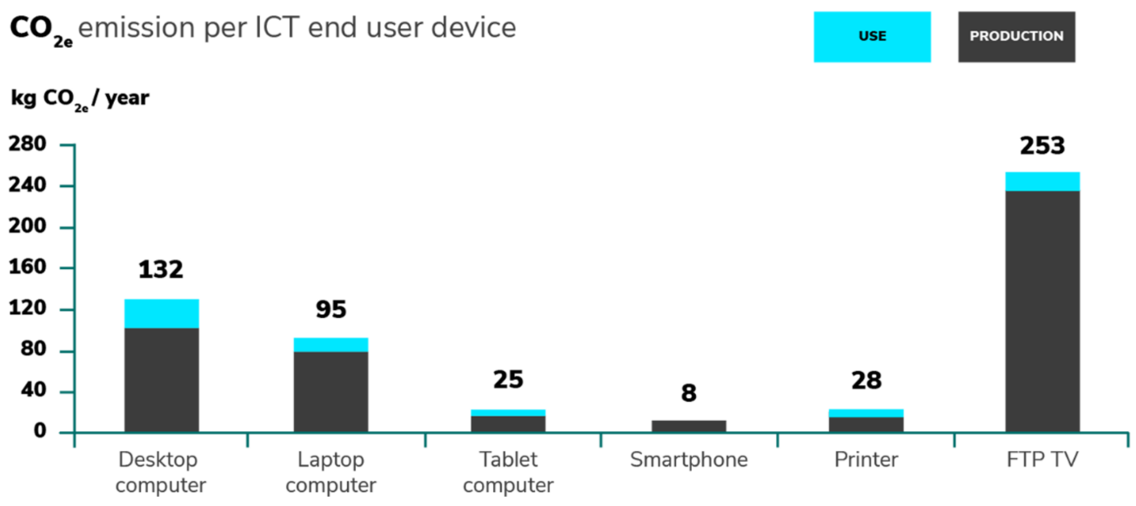


Figure 3. Emissions per ICT device [29] [30].

Figure *3* illustrates the ratio of emissions related to the usage and production of end user devices. Production is the main contributor to GHG emissions across all types of devices. These emissions are driven up by the utilization of different raw materials and complex manufacturing processes. The impact of production emissions can be reduced by extending hardware lifespan [30].

3.2 Extending lifespan

By the time a new device is purchased, it has already emitted a significant amount of pollution in the manufacture process. In addition to that, devices have a limited lifespan, which means that eventually, they will be unable to handle modern workloads and will require replacement. One approach to account for embodied carbon is to amortize the carbon over the expected life span of the device as illustrated in Figure *4*. [29]

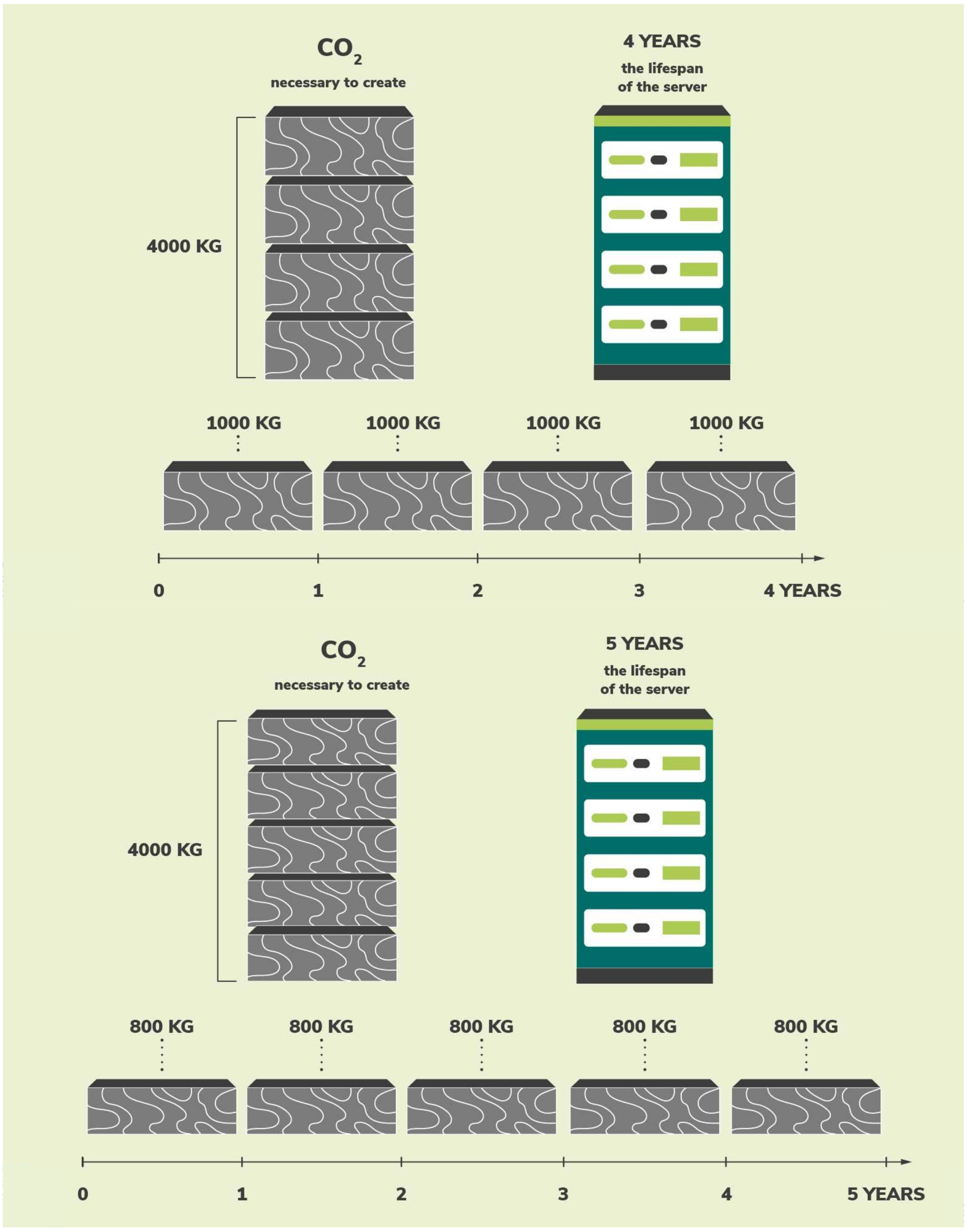


Figure 4. Increasing the usage time to reduce amortized carbon [29].

Hardware lifespan can be extended by increasing the usage time of the device. This can be achieved by developing software that runs on older hardware, and thereby reduce the *modern workload* [31]. Figure *4* illustrates how adding one year into the lifespan could reduce the yearly amortized carbon.

3.3 Static power draw & Energy proportionality

The **static power draw** of a computer means how much electricity is drawn in idle state. Computer is in idle state when it is not processing any tasks or performing any operations. Servers require power even when they are in idle state because they must remain powered up in order to maintain their state of readiness. This can mean for example, keeping the network connection active, which allows the server to be available to receive new tasks. The amount of power consumed in idle can vary depending on the server's hardware specifications, and the power management configurations [32].

The **energy proportionality** measures the relationship between power consumed by a computer and the rate at which it is being utilized [32]. Computer power consumption responds differently to varying utilization levels. This means that the energy proportionality of the computer is not linear, as illustrated in Figure *5*. Moreover, even energy-efficient servers tend to consume a relatively high amount of power when they are in idle state [33].

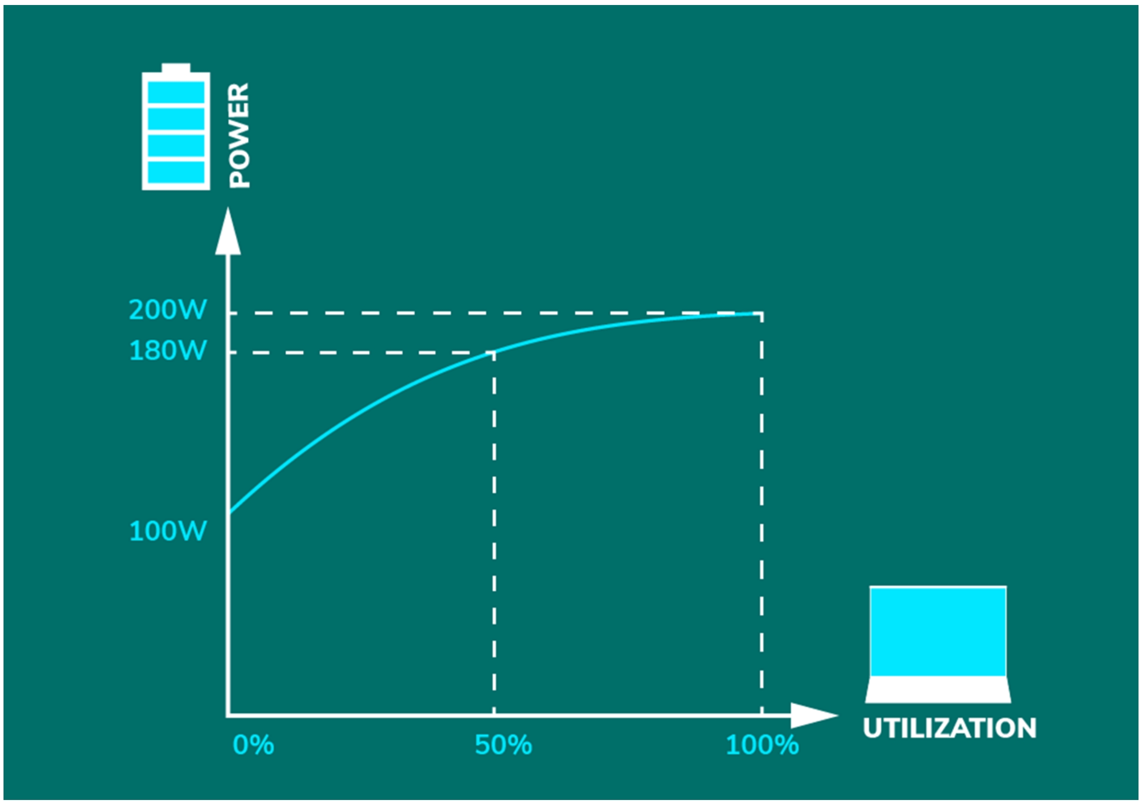


Figure 5. Energy Proportionality [32].

Because of this, the more a computer is utilized, the more efficiently it converts electricity to practical computing operations. However, a lot of resources recommend the exact opposite, and encourage developers to maximize idle (“race to idle”) [34] [35]. This is because hardware efficiency and embodied carbon are not taken into consideration in these materials. Hardware efficiency can be improved by Increasing utilization.

3.4 Increasing utilization

One way to increase hardware utilization and thus, improve hardware efficiency, is to run the workload on as few servers as possible, as illustrated in Figure *6*. Servers running at the highest utilization rate are maximizing energy efficiency and minimizing embodied carbon [29]. However, more research is required to determine how this affects the cooling demand and long-term durability of the device.

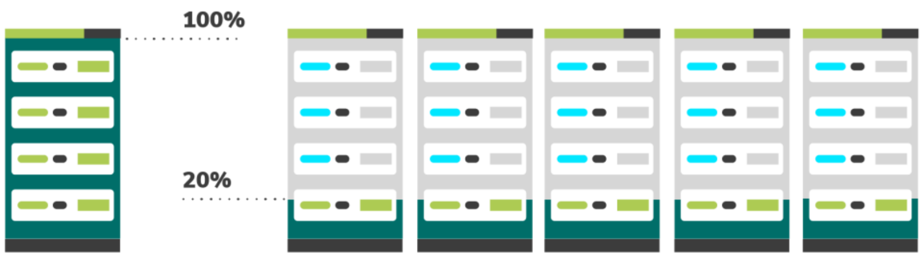


Figure 6. Increasing server utilization [29].

On the other hand, running servers at lower capacity ensures that peaks in demand can be handled without compromising performance [29]. Figure 7 shows how public cloud can offer high utilization rate and spare capacity.

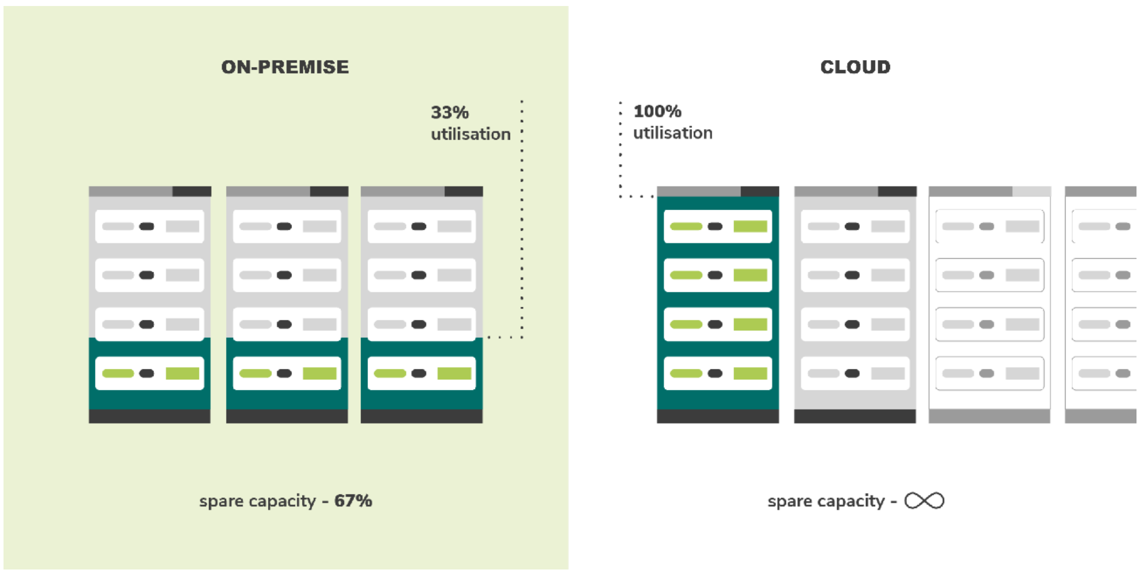


Figure 7. On-premise and cloud hardware utilization [29].

Cloud services typically increase the energy and hardware efficiency of the software application when compared to on-premise data centers. This is mainly due to reduced overheads and more efficiency in scalability [31] [36]. Therefore, moving computation to public cloud can reduce the overall emissions caused by the software application [29].

3.4.1 Virtualization

A typical system which is operated with only one operating system (OS) per device, generally has a relatively low usage rate with a lot of unused hardware resources, as illustrated in Figure 8. However, the introduction of virtualization technology has helped address the issue of low usage rates by making better use of the available resources of the equipment [37].

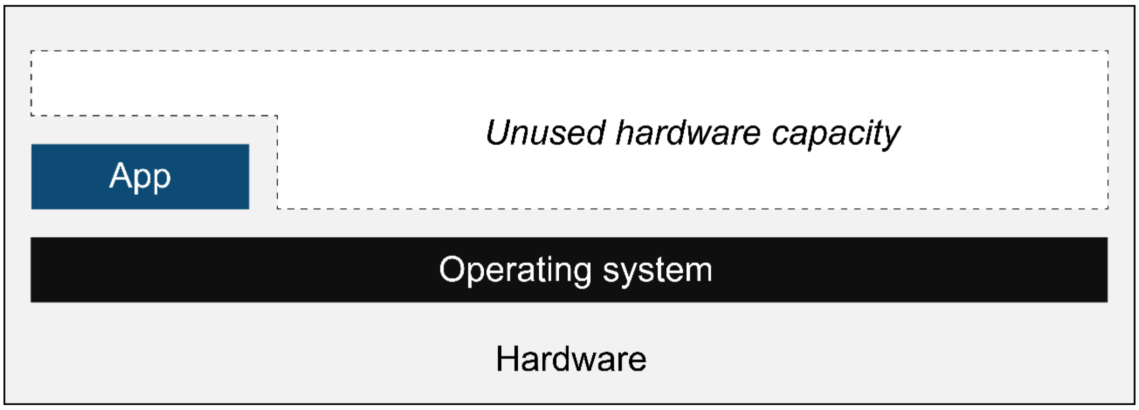


Figure 8. Example of a typical device.

Figure *8* shows an example where the device only contains a single application. This is often necessary, so that different applications can be **isolated** from each other. Isolation ensures that applications are running in a clean and safe environment. As different applications often require different dependencies, it is important to have control over resources and configurations [38].

Virtualization technology employs a hypervisor to abstract physical resources logically and allocate them to multiple operating systems running on a single device simultaneously [37]. Figure *9* visualizes how hypervisor allows multiple isolated applications to run in the same device while maximising hardware usage.

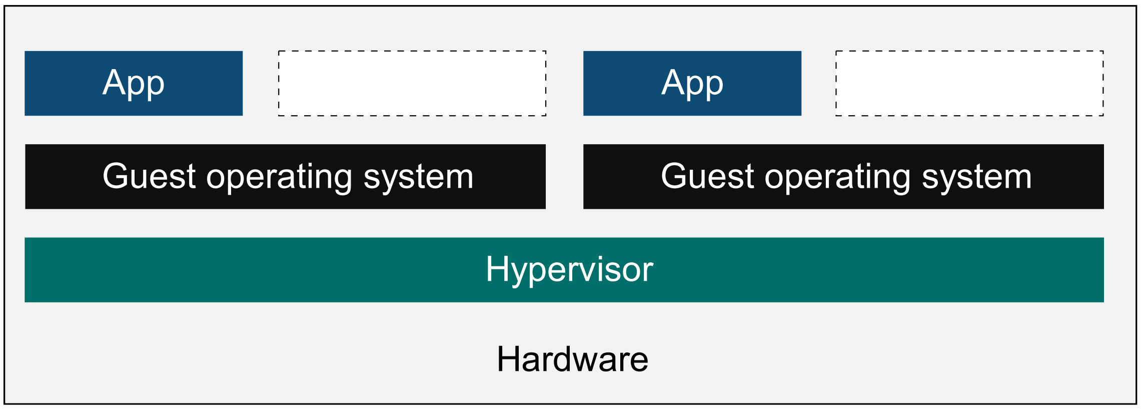


Figure 9. Example of virtualization.

Virtualized systems consume more energy than physical ones due to higher hardware utilization rate [39]. The energy consumption depends on the hypervisor used with KVM being the most efficient compared to Xen and vSphere [40].

3.4.2 Containerisation

Virtualization require the whole system (hardware, and OS) to be virtualized. However, with the use of containers, individual applications can be isolated from each other, and "virtual machines" can be created without the overhead associated with traditional virtualization setups. This means that containers allow for a more lightweight and efficient way to virtualize applications [41]. Containerization is illustrated in Figure *10*.

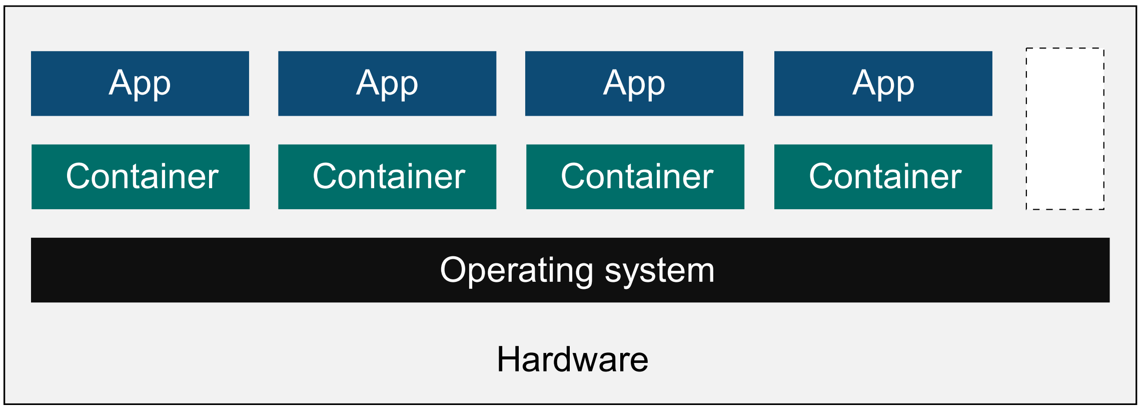


Figure 10. Example of containerization.

Containerization consumes less power compared to hypervisors [38] [42]. However, some cases of virtualization cannot be containerized, such as those where a different type of OS is required [41].

4. DEVOPS & PROJECT MANAGEMENT

Development and Operations (DevOps) combines a set of practices and tools that automate and streamline the software delivery process. This increases the organization’s ability to deliver applications and services more quickly and reliably [43]. This chapter goes over different factors that affect the energy consumption related to DevOps, including cloud, continuous integration and continuous deployment (CI/CD), and project management.

4.1 Cloud

The cloud provides services, such as computation, databases and servers over the internet. It allows accessing and utilizing these resources on-demand, without the need for local hardware or infrastructure. Although the energy consumption of the cloud is heavily affected by the cloud provider, there are some factors that can be taken into consideration by the cloud consumers.

4.1.1 Power Usage Effectiveness (PUE)

The power usage effectiveness (PUE) metric is widely used by data center industry, to measure the infrastructure energy efficiency for data centers. PUE is a measure of the energy used by computing equipment compared to the energy used for cooling and other overheads that support the equipment [44]. A data center with a PUE close to 1.0 means that computing is using almost all the energy, while a PUE of 2.0 indicates that an additional watt of power is required to cool and distribute power to the IT equipment for every watt of power used by the equipment.

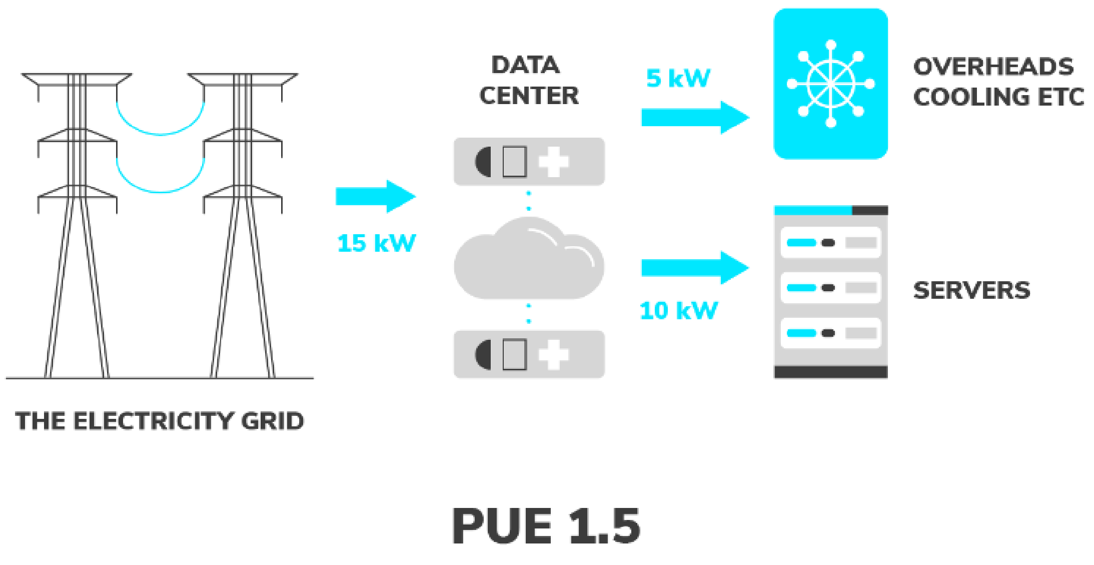


Figure 11. PUE 1.5 [32]

To put it simply, PUE can be thought of as a multiplier for an application's energy consumption. For instance, as illustrated in Figure 11, if an application consumes 10 kWh and the PUE of the data center it runs in is 1.5, the actual consumption from the grid is 15 kWh. This is because 5 kWh goes towards the operational overhead of the data center, while the remaining 10 kWh goes to the servers that run the application [32]. In 2021, the average PUE of data centers was 1.57 [45]. Choosing a cloud provider and data center with lower PUE can help increase the overall energy efficiency of the software.

4.1.2 Data center location

The location of the data center is an important factor of energy consumption. The further data travels, the more energy is consumed in transmitting the data through the network. Therefore, locating servers closer to users can help to reduce energy consumption. However, it can be hard to define the precise center of mass of the users, but website analytics can help to get a rough idea by identifying the country where core user groups are located [12].

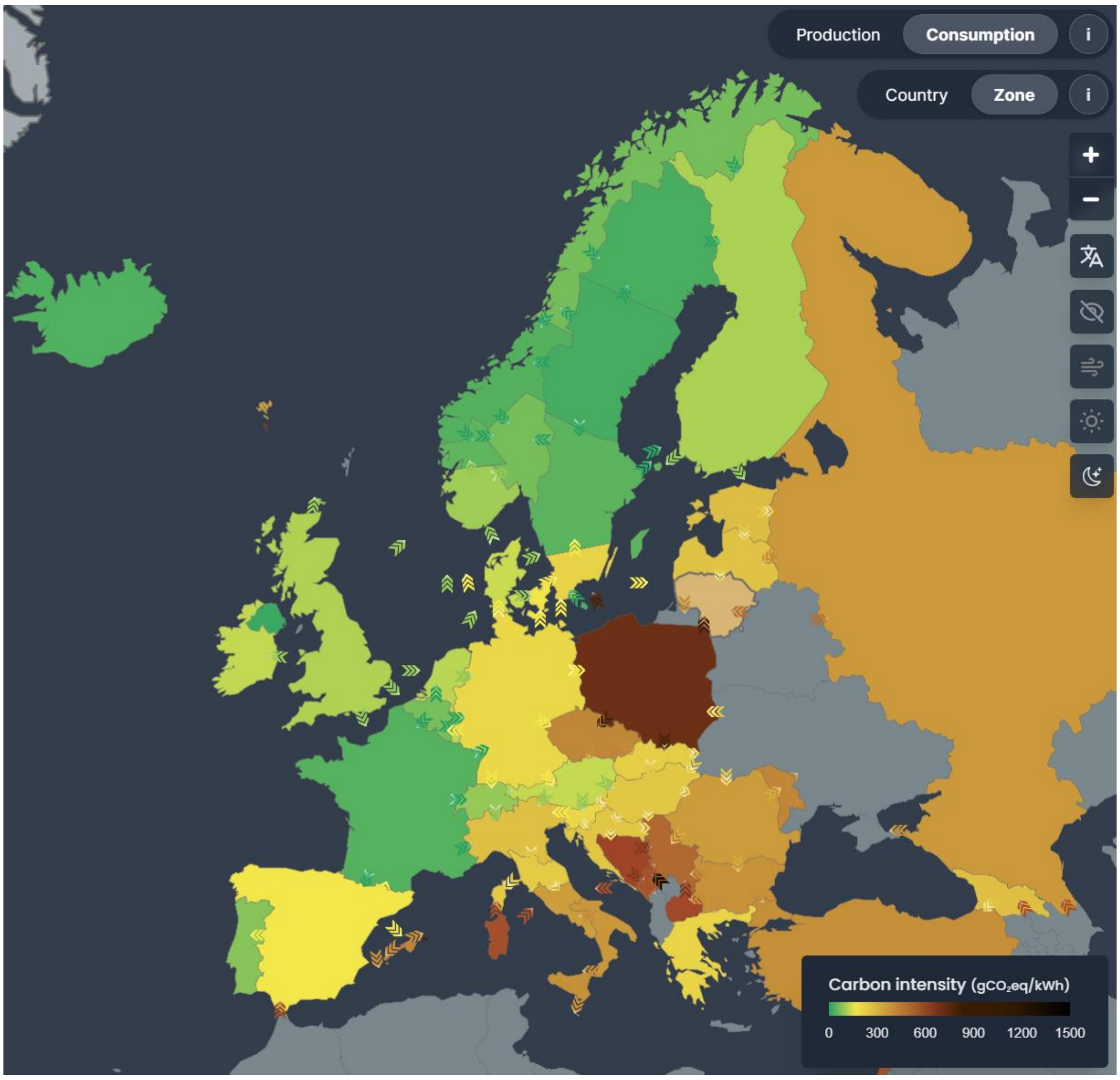


Figure 12. Live data of carbon intensity of electricity in Europe on November 19th 2023 [46].

The location of the data centre can also have an impact on the availability of clean energy sources for the data center. Figure *12* shows the electricity map that presents live data for the carbon intensity of electricity by country [46]. The usage of renewable energy sources also varies between different cloud providers. Major cloud providers, such as Microsoft, Google, and Amazon, are making investments in sustainable energy sources to power their data centers. Additionally, many providers engage in CO_2 certificate trading to present a more environmentally friendly image for the energy waste generated by their data centers [31]. Typically cloud providers are being very public about these kinds of commitments, which makes it easier to choose a more environmentally friendly provider.

4.2 CI/CD

Continuous Integration (CI) and Continuous Deployment (CD) is a software development approach that involves frequently integrating code changes into a shared repository, and automating the required processes to ensure that software is always in a releasable state [47]. The CI/CD process is automated in a pipeline, which can consist of various stages, such as code compilation, unit testing, integration testing, code analysing, packaging, and deployment.

Running the pipeline consumes energy and can take a very long time to complete [48]. As the pipeline typically includes very critical and essential operations that ensure software quality, such as testing, the energy cost of running the pipeline should be reduced without compromising software quality. This could be done by running the pipeline more strategically and avoiding unnecessary triggering of the process [49]. The frequency of running the pipelines and the workload of each run should be considered to reduce the energy consumption related to the CI/CD process.

To reduce the frequency of running the process, automatic triggers of the pipeline could be limited to the most important parts of the integration. For example, the pipeline could run automatically only when a complete feature (e.g., branch) is integrated, instead of when a single change (e.g., commit) is made. The pipeline could be triggered manually for individual changes if needed [48].

Additionally, the pipeline could run only partially in some situations. For example, tests could be run when a feature is ready without building and deploying the whole software, and unit tests could be skipped for units that have not changed. Configuring which parts of the pipeline are set to run can reduce the workload of the run. Alternatively, the CI/CD process could be skipped entirely for changes that are unlikely to impact any functionality of the software, such as changes made to the documentation or comments [49].

4.3 Methodologies

The choice of software development methodology can influence the environmental impact of the project. Different methodologies offer different approaches for planning, organizing, and managing the software development process. By choosing a method that optimizes resource utilization and reduces waste, the overall environmental sustainability of the software can be improved [50].

The **waterfall methodology** is a traditional project management approach that follows a sequential and linear process, where each phase of the project is completed before moving on to the next one [11]. The typical phases in waterfall methodology include requirements gathering, design, implementation, testing, deployment, and maintenance as presented in Figure *13*.

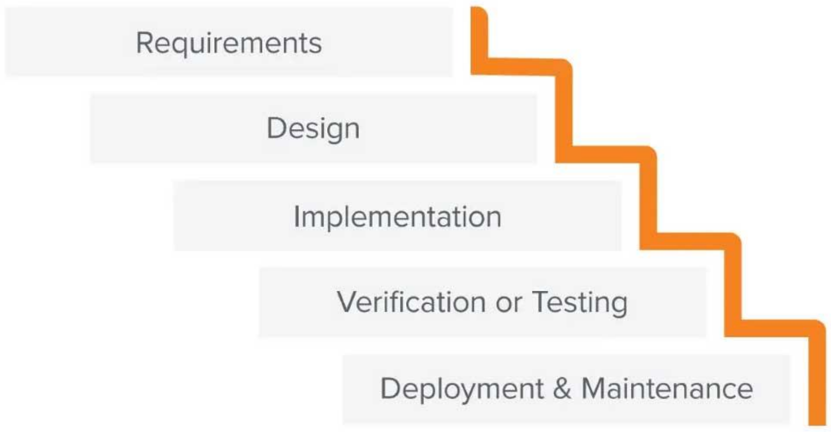


Figure 13. Example of waterfall methodology [51].

The waterfall methodology can promote waste in several ways through wasted efforts and profitless work. It involves a significant amount of upfront planning, documentation, and design work, which does not account for new ideas and knowledge that arise during the development process. This also makes it challenging to adapt or incorporate changes in requirements efficiently [11]. Furthermore, there is rather limited communication and collaboration between team members and stakeholders during the development process. The lack of regular feedback can result in misunderstandings and thereby, wasted efforts [11].

The **agile methodology**, on the other hand, is a flexible and iterative approach to software development that emphasizes continuous collaboration between developers and stakeholders. Agile methodology focuses on iterative and incremental development, which means that the development team delivers working software in short cycles which is visualized in Figure *14* [50].

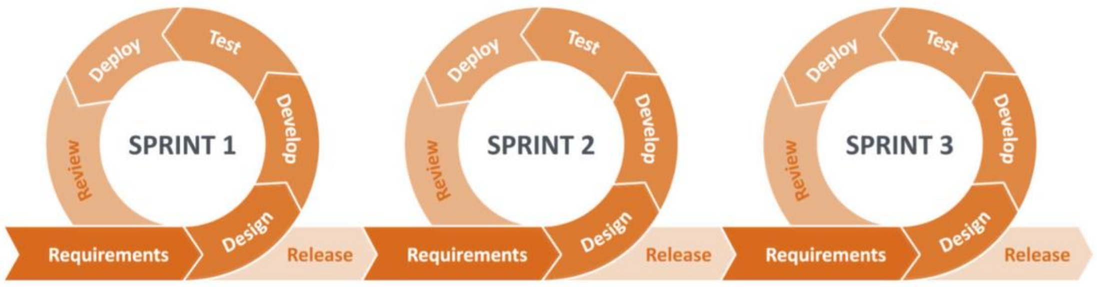


Figure 14. Example of agile methodology [52].

By developing and delivering software in small increments that are frequently tested, less time and resources are wasted, since developers can quickly identify and fix problems as they arise. Agile methodology also involves continuous feedback loops between developers and stakeholders, which allows the development team to identify areas of improvement and quickly react to changes in requirements [11]. Choosing agile methods can help utilize resources more efficiently and reduce wasted time and effort which improves the overall environmental sustainability of the software [53].

5. WEB APPLICATION DEVELOPMENT

This chapter describes how different aspects affect the energy consumption of web applications. Web application development includes front-end, back-end and mobile development. Each section goes over various factors that affect the energy consumption and introduces ways to improve these.

5.1 Front-End Development

Front-end refers to the client, which is the user-facing part of a software application. It includes the visual and interactive elements that users see and interact with directly.

Modern web applications are typically implemented as a **single-page application** (SPA). A single-page application is a web application that dynamically updates the content on a single web page, instead of loading separate pages for different interactions. SPAs use JavaScript to retrieve and display data without requiring a full page reload, resulting in a faster and more interactive user experience [54]. However, this typically requires extensive client-side JavaScript processing, which can be energy intensive.

Server-side rendering (SSR) is a technique, where the server renders the web page and sends the HTML to the client, instead of relying on client-side JavaScript to render content. This approach reduces the client-side processing required, resulting in faster initial load times and potentially lower energy consumption on the client's device [55]. SSR is suitable for pages that consist of static content as frequent content updates can lead to increased data traffic.

Static Site Generation (SSG) is another technique for rendering web pages on the server-side. When the data required to render a page is consistent for all users, the page can be rendered only once during the build process, instead of rendering the page for each client separately on every request. SSG can only be applied to pages that use static data, that is known during the build process and remains unchanged between deployments. Any updates to the data require a new deployment [56].

5.1.1 DOM

Document Object Model (DOM) is the browser's way of conceptualizing document content by turning the web page code into visual objects. Each HTML element is represented as a node in the DOM. The number of nodes in the DOM determines how much memory the browser requires and the time it takes to display and update the page. A higher number of nodes in the HTML means more time spent processing and rendering each element. Moreover, any interaction with the DOM through JavaScript requires additional processing time and memory to navigate the DOM elements [57] [58].

Different frameworks use different approaches to manipulate the DOM. Some frameworks, like **React** and **Vue**, use a virtual DOM to efficiently update the actual DOM. They update a virtual representation of the DOM first and compare it to the actual DOM. This allows the browser to only re-render the parts that have changed, minimizing actual DOM manipulations, and reducing the computational workload. Other frameworks, like **Angular**, re-rendering the entire DOM whenever there is a change in the application state [59]. This can lead to more extensive and potentially less energy-efficient DOM manipulations and updates.

5.1.2 Frameworks

In addition to DOM manipulation, there are several other aspects that can affect the efficiency of a front-end framework, such as bundle size and the number of components and elements within them, that need to be processed on each render [60] [59]. These are listed for three of the most popular frameworks in Table *3*.

Table 3. Front-end frameworks. [59]

Framework	Bundle size	DOM	Components processed	Elements processed
Angular	Large	Real	All	Bindings only
React	Small	Virtual	Subtree of updated component	All
Vue	Small	Virtual	Dirty components only	Bindings only

Of these frameworks, Vue is the most efficient choice, showing very similar performance to vanilla JavaScript. It is followed by Angular, which was more performant than React

[59] [60] [61]. However, React was found the most responsive in terms of being ready to accept user interactions after DOM manipulation [60]. Although the performance was measured through duration, it correlates with energy consumption, since the tests were run on the same processor at full power. It is also worth noting that there are more efficient versions of React available, such as Preact, which shows similar results to Angular or even Vue in some cases [61].

5.1.3 WebAssembly

For over two decades, JavaScript has been the standard scripting language for client-side web development. However, it was not designed with performance in mind, and it falls short in energy efficiency. To overcome this, a new portable and efficient bytecode language, WebAssembly (Wasm), has been developed [62]. It is a low-level assembly-like language with a compact binary format [63]. Assembly is human-readable abstraction on top of machine code.

Programs written in other languages can be run in the browser by compiling them to Wasm. This way the performance intensive JavaScript code can be replaced with more energy efficient languages [63]. This has been shown to have a significant effect on the energy efficiency [64].

5.2 Back-End Development

Back-end refers to the server-side components of a software application. It is responsible for handling tasks such as data storage, business logic, and communication with the front-end. In a typical scenario, the server receives requests from the front-end, processes them, retrieves, or manipulates data from databases, and sends back the appropriate response.

5.2.1 HTTP

The server typically communicates with front-end via Hyper Text Transfer Protocol (HTTP). This HTTP communication is typically the most energy consuming operation of the network [65]. One way to reduce energy consumption related to HTTP communication is by bundling multiple small HTTP requests into fewer, larger requests [66].

Another technique involves attempting to parse an HTTP request with the assumption that it is valid and aborting if the parsing fails, rather than first validating that it is parseable and then parsing it. Parsing gives access to the validated information straight away, while validation throws it away and then it needs to be parsed anyway [67].

5.2.2 Database

Using efficient database queries and minimizing unnecessary data retrieval can help reduce the energy consumption of a software application. To reduce the amount of data needed to transfer, data-centric calculations can be performed in the database [68]. This can mean for example, querying only a certain selection of the data, instead of querying all the data and filtering it in the application code.

More complex calculations and CPU (Central Processing Unit) intensive operations are more efficient to do in the application code, rather than in the database [69]. However, aggregate functions, such as MIN, MAX, SUM, and AVG, are typically more performative than the equivalent code implementation [68].

Moreover, the type of the database can have an impact on the energy efficiency of the application. Traditional relational databases, such as MySQL or PostgreSQL, offer support for structured data with complex relationships. However, they commonly act as a bottleneck withing an otherwise parallel application, in which case delegating some of the processing away from the database might be beneficial [68].

On the other hand, NoSQL databases, such as MongoDB or Cassandra, excel at handling large volumes of unstructured data with high scalability requirements. They are more efficient at retrieving and storing data than relational databases [70] [71]. However, the use of NoSQL databases might increase the need for data processing in the application code, which could end up being less energy efficient. This needs further research.

5.2.3 Frameworks

Back-end frameworks have significant differences in their efficiencies [72]. Lightweight frameworks (e.g., Fastify, FastAPI), that have minimal overhead and dependencies, tend to consume fewer server resources, resulting in lower energy consumption compared to heavier alternatives (e.g., Express, Django). Figure *15* shows how many requests some of the most popular frameworks can process in a second (higher is more efficient).

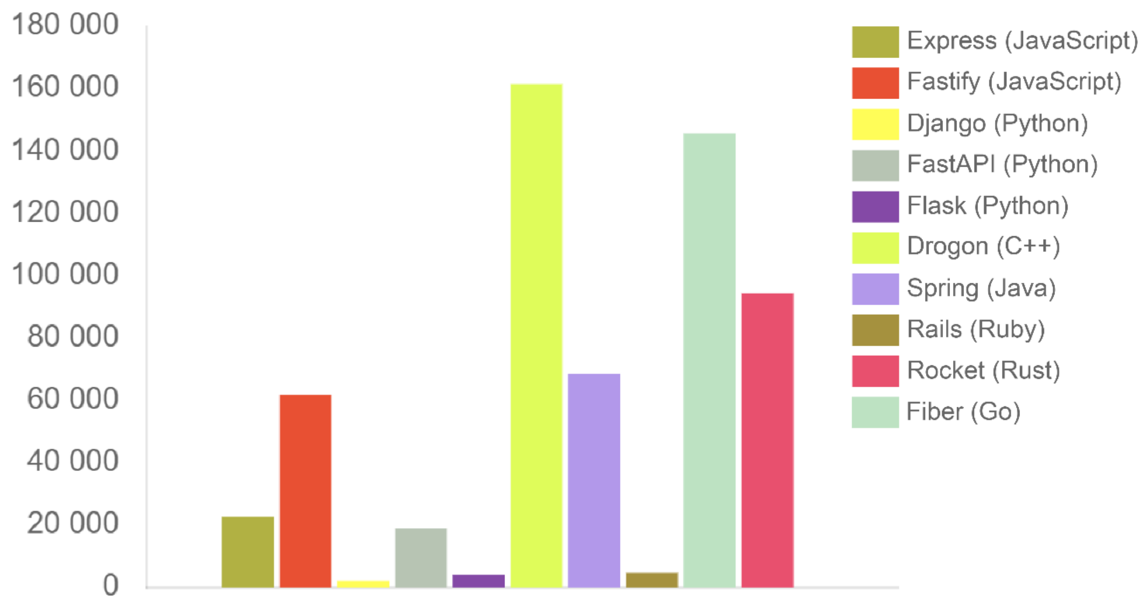


Figure 15. Requests per second for different Back-end frameworks (2023) [73].

The choice of programming language heavily affects the performance, but there are also notable differences in frameworks within the same language. For example, Fastify can process nearly three times more requests per second compared to Express, even though they are both JavaScript frameworks. Similarly with Python frameworks, Flask can process twice as much as Django, and FastAPI can process over three times more than Flask. [73]

5.3 Mobile Development

Mobile devices such as smartphones and tablets derive the required energy from batteries, which have a limited size and capacity. Therefore, it is crucial to manage energy consumption effectively [65]. Figure *16* shows how energy consumption is distributed in a typical mobile device. However, it is worth noting that this can vary a lot based on the activity and device used [74].

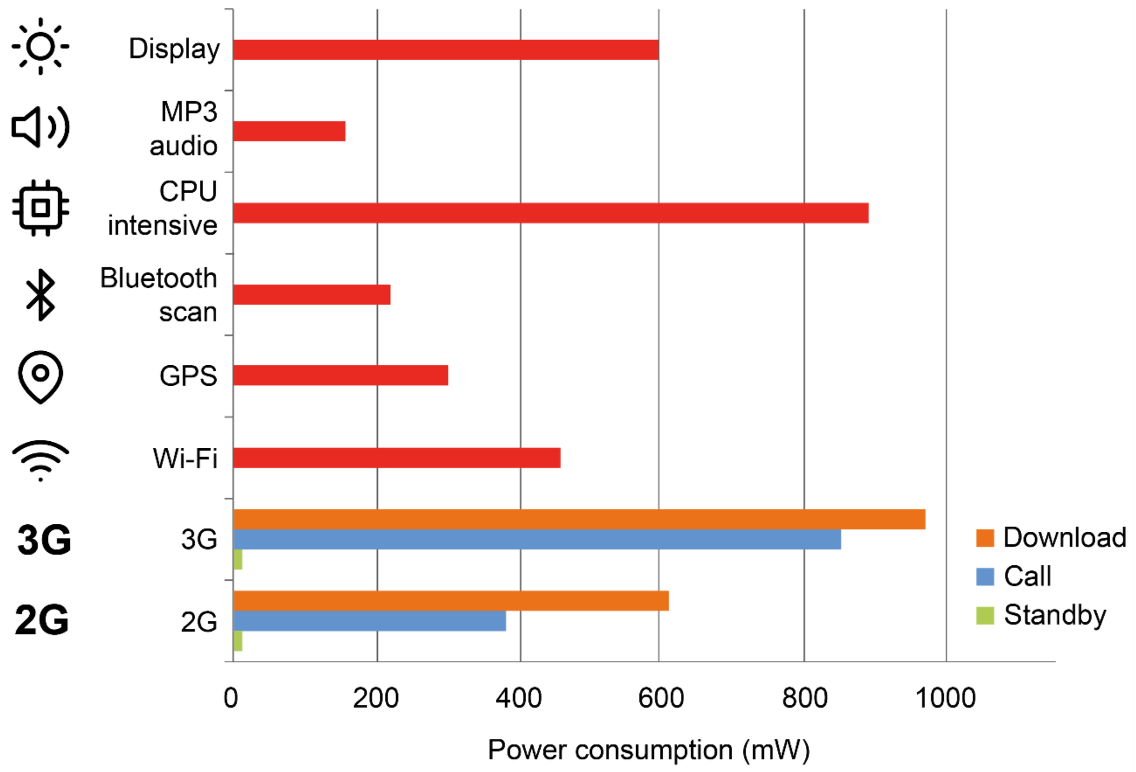


Figure 16. Power consumption distribution in a mobile device (2013) [3].

As seen from Figure *16*, the **CPU** is one of the most power consuming components. Some resources suggest CPU offloading to reduce energy consumption. This means moving computation to an external execution environment (e.g., the cloud) [75] [76]. However, although moving energy consumption to a different location can reduce the consumption within the device, the effect on the total energy consumption requires more research.

Mobile applications can often include background activities, which refer to the execution of tasks or operations that continue to run even when the application is not actively in the foreground or visible to the user [77]. Background activities enhance the user experience, improve app functionality, and provide timely updates and notifications. However, they can consume a lot of energy as they require the device to be activated or “woken up”. This could be optimized by bundling activities together, instead of activating the device multiple times, as illustrated in Figure *17* [78].

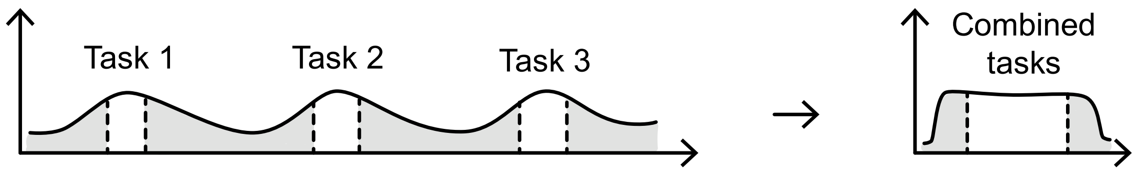


Figure 17. Combining background activities to optimize the activation energy needed [78].

The **display** is also one of the most power consuming components of the device. Therefore, limiting the screen time can be an effective way to reduce the power consumption. This can be done, for example, by allowing users to interact with the application using alternative interfaces, such as audio or earphone buttons, although more research is needed to evaluate the power consumption of such alternative interfaces. The power consumption of the screen can also be affected by user interface design as discussed in Chapter 7 [78].

Another highly power consuming component is the **cellular network**. As Wi-Fi consumes significantly less energy, it is advised to use that instead of cellular. Heavy data connections could be delayed or disabled until the device is connected to Wi-Fi. [78]

Additionally, there are several approaches to consider when **developing applications for multiple platforms**, such as Android and iOS. One option is to develop applications natively, which requires separate applications to be implemented for each platform. This is the most energy efficient approach, however, building and maintaining several code bases with different technologies requires additional work and knowledge from the development team [79].

Another option is to use cross-platform development approaches, such as Flutter, Capacitor, React Native or progressive web applications (PWA). These allow apps to be deployed for multiple platforms from a single code base. From these approaches, Flutter has been shown to be the most energy efficient choice, followed by Capacitor and PWAs, with React Native being the least efficient [79].

6. SOFTWARE IMPLEMENTATION

Software is getting slower at a faster rate than hardware is becoming faster. This means that the improvements made in microchips through electrical engineering is being outpaced by the software utilizing the hardware [80]. Software engineers are typically not required to take energy efficiency into consideration during the development process [81]. This chapter goes over different aspects of software implementation and the energy optimization possibilities related to these.

6.1 Architecture

A well-designed software architecture can significantly help decrease energy consumption. Research has shown that object-oriented programming (OOP) that implements metrics of high-quality architecture, such as modularity, scalability, and reusability, can reduce energy consumption by up to 30% [82]. These metrics are described in Table *4*.

Table 4. Architectural metrics. [82]

Metric	Description
Loose coupling	Components do not depend on each other and can operate independently.
Abstraction of communication	Hiding away communication details behind a simple interface.
Expressive power	Simplicity of the architecture so that it can be understood easily.
Evolutionary power	Easiness of updating and extending the software in the future.
Depth of packages	The depth of subpackages used for composition.

Modularity refers to the practice of breaking a system down into smaller, reusable units (modules) that can be developed, tested, and maintained independently. Modular software consists of more components, which increases the communication between them but reduces the payload on each message. In the context of database access, modularity has been shown to decrease energy consumption. While the software consisted of more database calls, the reduced amount of data transferred per call resulted in less overall energy consumption. [83]

However, modularity can lead to **software bloat**, which refers to software becoming larger and heavier, affecting the performance and energy efficiency of the program. Modularity itself is not the issue, but as modules are often designed to be as generic as possible, they are harder to optimize for specific use cases. In doing so, programmers unintentionally introduce unnecessary processing and data overhead for the sake of reusability. [84]

Another thing that can cause software bloat and increase energy consumption is the excessive use of external **libraries** [85]. Libraries can often be very bloated, and they might come with a lot of unnecessary components. For example, a library for drawing diagrams can consist of multiple different types of diagrams, when only one specific type of diagram is used. However, libraries can also be very modular in which case, by being selective about which parts to include, file sizes can be kept minimal.

The **high-level architecture** dictates the workload distribution of the software application. For web applications, moving computation away from the end-user devices (client) can help increase the overall energy efficiency of the software, assuming computation is more energy efficient in another environment [14]. This can be achieved through various architectural patterns.

In **Client-Server Architecture**, which is visualized in Figure 18 the server is responsible for processing and managing data, while the client mainly handles the user interface. This allows multiple clients to access the same server where common calculations can be executed once instead of every client performing the calculations themselves [86].

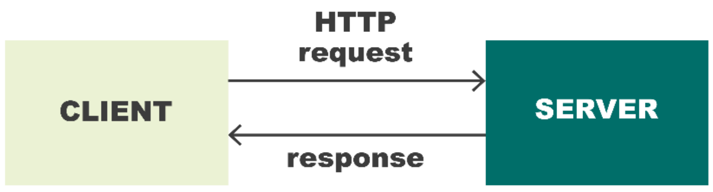


Figure 18. Client-server architecture.

Microservices Architecture extends the Client-Server Architecture by dividing the server-side responsibilities into smaller, independent services that communicate with each other through APIs (Application Programming Interface). This is illustrated in Figure

19. Different services can be responsible for e.g., storing data, running complex calculations, and accessing data from an external source [86].

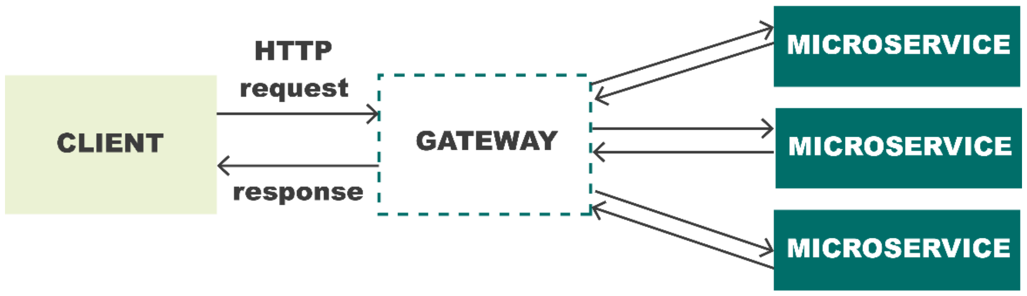


Figure 19. Microservices architecture.

This allows services to be deployed independently and scaled up or down as needed, which can reduce energy consumption [87]. However, it typically requires more frequent API calls, which can increase energy consumption on the network [65]. Further research is required to compare the energy efficiencies of these architectural patterns.

6.2 Programming language

Programming languages differ in many mechanisms that can affect energy efficiency. One such significant mechanism is the **execution type** of a programming language, which defines how code written in that language is processed and run. Languages can be compiled, which means that the code is translated to machine code before execution. Interpreted languages are translated and executed simultaneously one instruction at a time. In virtual machine languages, code is translated to an intermediate form before it can be executed by a virtual machine [88]. Table *5* presents the energy efficiencies of different programming languages.

Table 5. Programming language efficiencies [89].

Rank	Language	Execution type	Energy (J)
1	**C**	Compiled	1.00
2	**Rust**	Compiled	1.03
3	**C++**	Compiled	1.34
4	**Ada**	Compiled	1.70
5	**Java**	Virtual machine	1.98
6	**Pascal**	Compiled	2.14
7	**Chapel**	Compiled	2.18
8	**Lisp**	Virtual machine	2.27
9	**Ocaml**	Compiled	2.40
10	**Fortran**	Compiled	2.52
11	**Swift**	Compiled	2.79
12	**Haskell**	Compiled	3.10
13	**C#**	Virtual machine	3.14
14	**Go**	Compiled	3.23
15	**Dart**	Interpreted	3.83
16	**F#**	Virtual machine	4.13
17	**JavaScript**	Interpreted	4.45
18	**Racket**	Virtual machine	7.91
19	**TypeScript**	Interpreted	21.50
20	**Hack**	Interpreted	24.02
21	**PHP**	Interpreted	29.30
22	**Erlang**	Virtual machine	42.23
23	**Lua**	Interpreted	45.98
24	**Jruby**	Interpreted	46.54
25	**Ruby**	Interpreted	69.91
26	**Python**	Interpreted	75.88
27	**Perl**	Interpreted	79.58

As Table *5* shows, compiled languages tend to consume the least energy, interpreted languages consume the most, and virtual machine languages are scattered in the middle. Interpreted languages are typically more readable, easier to learn, and faster to develop. The trade-off between efficiency and easiness should be taken into consideration when choosing a programming language.

It is worth noting, that there have been some discussions [90] [91] regarding the results of [89], and particularly the energy consumption of TypeScript. Since TypeScript compiles to JavaScript, it should be expected to show more similar results. In the study, one test (fannkuch-redux) results TypeScript to consume about 1000 times more energy compared to JavaScript, as opposed to the other tests, which only resulted in minor differences. This has been speculated to be an implementation error in the test code, which is available in GitHub [92]. In the source code, the JavaScript and TypeScript implementations of the fannkuch-redux are somewhat different, and therefore this matter needs further research. However overall, the study is indicative.

Table *6* shows the estimated popularities of different programming languages. The popularity has been estimated by analyzing how often language tutorials are searched on Google [93].

Table 6. Popularity of programming languages [93].

Rank	Languge	Change	Share	Trend
1	**Python**	-	27.43 %	-0.2 %
2	**Java**	-	16.19 %	-1.0 %
3	**JavaScript**	-	9.4 %	-0.1 %
4	**C#**	-	6.77 %	-0.3 %
5	**C/C++**	-	6.44 %	+0.2 %
6	**PHP**	-	5.03 %	-0.4 %
7	**R**	-	4.45 %	+0.1 %
8	**TypeScript**	-	3.02 %	+0.3 %
9	**Swift**	↑	2.42 %	+0.4 %
10	**Rust**	↑↑↑	2.15 %	+0.6 %
11	**Objective-C**	↓↓	2.13 %	+0.0 %
12	**Go**	↓	2.01 %	+0.0 %
13	**Kotlin**	↓	1.79 %	+0.0 %
14	**Matlab**	-	1.59 %	+0.0 %
15	**Ruby**	-	1.1 %	-0.0 %
16	**Ada**	↑↑↑↑	1.06 %	+0.3 %
17	**Powershell**	↑	1.06 %	+0.2 %
18	**VBA**	↓↓	0.91 %	-0.1 %
19	**Dart**	↓↓	0.86 %	-0.0 %
20	**Lua**	↑↑	0.64 %	+0.0 %
21	**Visual Basic**	-	0.58 %	-0.0 %
22	**Abap**	↑↑	0.57 %	+0.1 %
23	**Scala**	↓↓↓↓	0.57 %	-0.2 %
24	**Julia**	↓	0.42 %	-0.1 %
25	**Groovy**	-	0.42 %	-0.0 %
26	**Haskell**	↑	0.3 %	+0.0 %
27	**Perl**	↓	0.29 %	-0.0 %

As seen in Table *6*, Python is the most popular language despite being one of the least efficient. However, it delegates a lot of the workload to libraries written in other, more efficient languages, which makes Python programs perform more efficiently [94] [95] [96]. This practice is used in some other languages as well, such as JavaScript (V8, Node) [97] [98].

6.3 Concurrency

Concurrent programming is a standard practice in software engineering [99]. It allows a program to execute multiple tasks or processes simultaneously, without waiting for one task to complete before starting another [100]. This can be achieved through various techniques such as multi-threading, multi-processing, or distributed computing.

Multithreading can reduce energy consumption, assuming the workload can be split evenly [101]. However, forcing algorithms to run in parallel might have the contrary effect, which is why multithreading should be considered on a case-by-case basis [99].

6.4 Data Structures & Algorithms

Many programmers have a tendency to select their preferred data structures, often without taking performance or energy consumption into consideration or knowing that there could be more optimal alternatives [102]. The **Big-O notation** is commonly used to analyze the efficiencies of data structures and algorithms. It represents the worst-case complexity, while the average complexity is represented by Big Theta (Θ), and the best-case is represented by Big Omega (Ω) [103] [104]. In the Big-O notation, *n* represents the number of elements, which can be for example the number of items in an array [103]. Figure *20* illustrates some common Big-O notations.

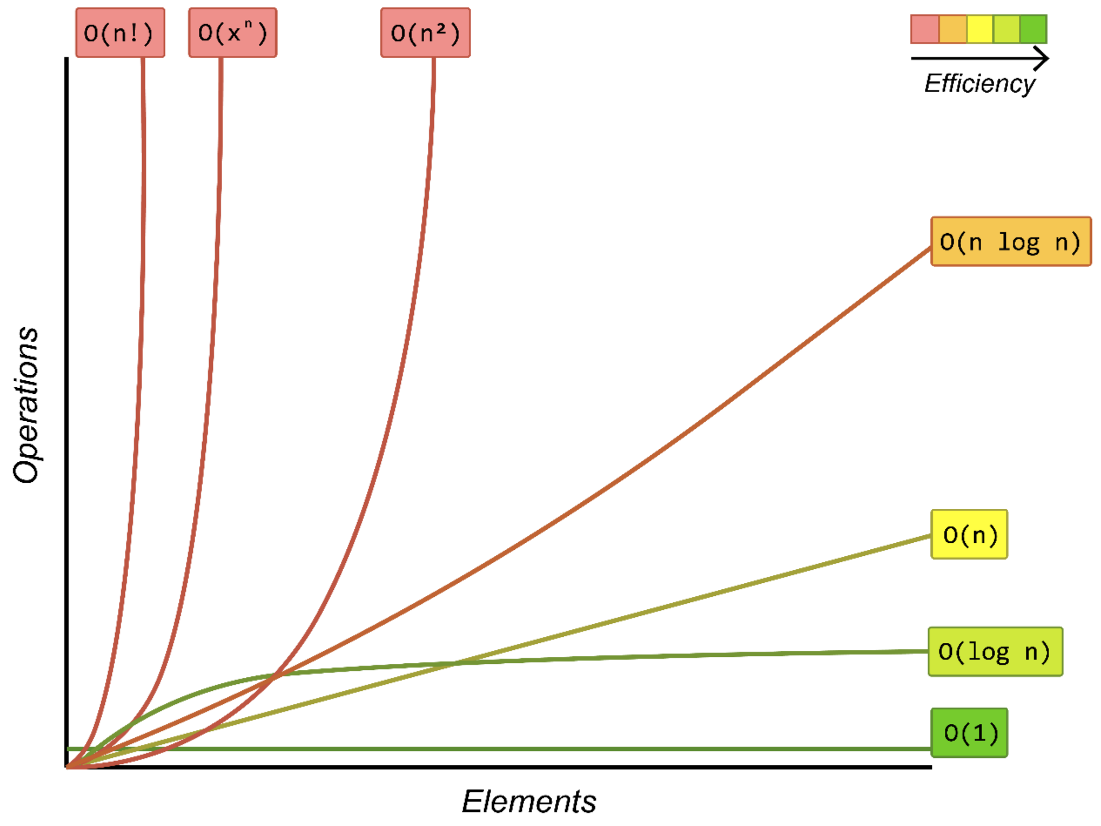
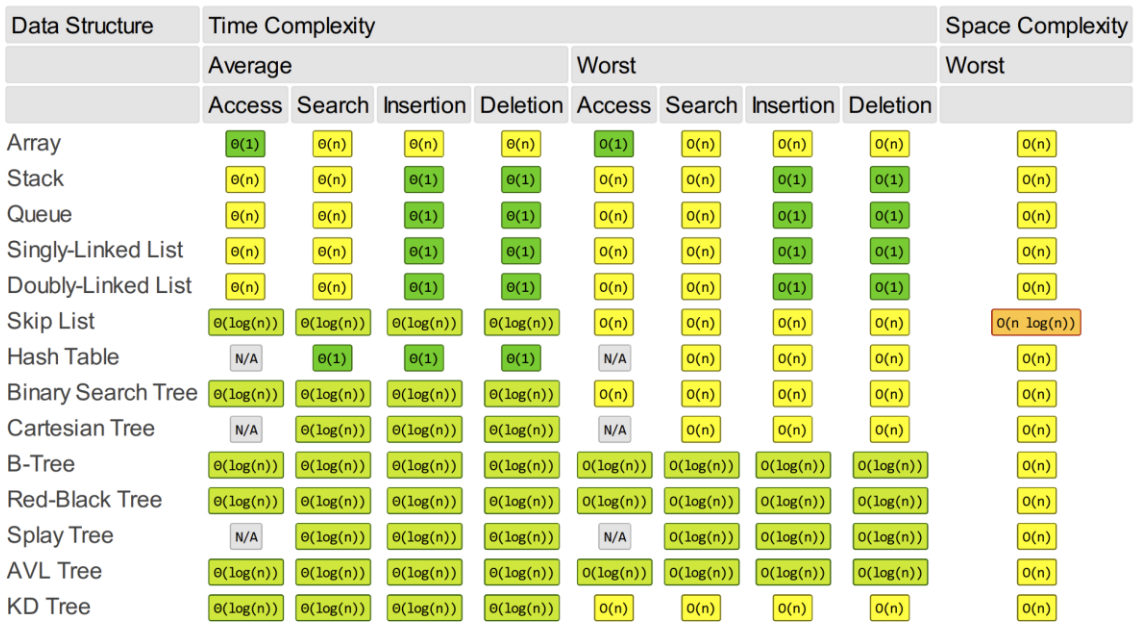


Figure 20. Big-O notation [105].

Figure *21* presents the efficiencies of commonly used data structures. Different programming languages often have distinct names for the same data structures. For example, the default hash table implementation is named Map in JavaScript, unordered_map in C++, and dict in Python [106] [107]. In contrast, a map in C++ is typically implemented as a Binary Search Tree or a Red-Black Tree [108] [109].

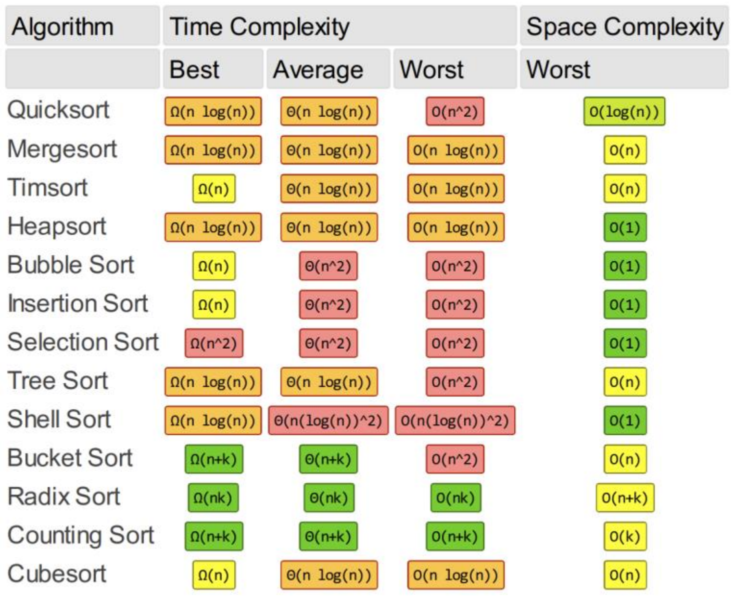


Data Structure	Time Complexity								Space Complexity
	Average				Worst				Worst
	Access	Search	Insertion	Deletion	Access	Search	Insertion	Deletion	
Array	Θ(1)	Θ(n)	Θ(n)	Θ(n)	O(1)	O(n)	O(n)	O(n)	O(n)
Stack	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Queue	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Singly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Doubly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Skip List	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n log(n))
Hash Table	N/A	Θ(1)	Θ(1)	Θ(1)	N/A	O(n)	O(n)	O(n)	O(n)
Binary Search Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)
Cartesian Tree	N/A	Θ(log(n))	Θ(log(n))	Θ(log(n))	N/A	O(n)	O(n)	O(n)	O(n)
B-Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Red-Black Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Splay Tree	N/A	Θ(log(n))	Θ(log(n))	Θ(log(n))	N/A	O(log(n))	O(log(n))	O(log(n))	O(n)
AVL Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
KD Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)

Figure 21. Common data structure operations [105].

When choosing which data structure to use, it is important to consider what are the most common operations for the data and choose a structure that is particularly efficient in those. However, also the circumstances and absolute efficiencies of these operations should be taken into consideration. For example, even though their average notations are the same, access operation in an array is more efficient than search operation in a hash table.

Also, the algorithm choice can have a significant impact on energy consumption [110].



Algorithm	Time Complexity			Space Complexity
	Best	Average	Worst	Worst
Quicksort	Ω(n log(n))	Θ(n log(n))	O(n^2)	O(log(n))
Mergesort	Ω(n log(n))	Θ(n log(n))	O(n log(n))	O(n)
Timsort	Ω(n)	Θ(n log(n))	O(n log(n))	O(n)
Heapsort	Ω(n log(n))	Θ(n log(n))	O(n log(n))	O(1)
Bubble Sort	Ω(n)	Θ(n^2)	O(n^2)	O(1)
Insertion Sort	Ω(n)	Θ(n^2)	O(n^2)	O(1)
Selection Sort	Ω(n^2)	Θ(n^2)	O(n^2)	O(1)
Tree Sort	Ω(n log(n))	Θ(n log(n))	O(n^2)	O(n)
Shell Sort	Ω(n log(n))	Θ(n(log(n))^2)	O(n(log(n))^2)	O(1)
Bucket Sort	Ω(n+k)	Θ(n+k)	O(n^2)	O(n)
Radix Sort	Ω(nk)	Θ(nk)	O(nk)	O(n+k)
Counting Sort	Ω(n+k)	Θ(n+k)	O(n+k)	O(k)
Cubesort	Ω(n)	Θ(n log(n))	O(n log(n))	O(n)

Figure 22. Array sorting algorithms [105].

Figure *22* shows, how sorting an array can vary in performance. This applies to other algorithms as well. Different algorithms might result in the same outcome, but their efficiency can differ significantly. Decreasing algorithmic complexity can be an effective way to reduce energy consumption [34]. In other words, focus on improving the mathematical algorithm instead of the code implementation.

6.5 Cache

Avoiding unnecessary processing is an effective way to reduce the energy consumption of a software. It is very typical, that data is processed in real time, even if the content is not real-time, and that the same calculations are performed multiple times [111]. This can be avoided by caching. **Cache** is a temporary storage location that stores frequently accessed data.

Caching can be done on multiple levels of the software application. Different cache mechanisms can be included for example, in code-level calculations or in communication. **Code-level** caching can be for example, storing the result of a function call to a variable so the value can be reused in other calculations. Additionally, some languages, such as Python, have built-in tools for caching [112].

The **communication** between client and server also offers many opportunities for caching. Results can be cached by the requesting or by the responding party. The requesting party (e.g., client) can cache the received response, so that it does not need to request it again [78]. The responding party (e.g., server) can cache the result so that it does not have to calculate it again. Similarly, the server can cache the received data from a database.

Moreover, HTTP supports caching with *Cache-Control* and *Expires* headers. These allow the client to fetch updated information after an expiration period or ask if an update is needed [113].

Although caching can effectively reduce the energy consumption of a software, it is worth noting that not everything can be cached, as sometimes real-time data is needed. Caching can be used when the same calculations, operations, or communication are performed repeatedly.

6.6 Synchronization

In some cases, software components may need to constantly synchronize or poll for updates, which can lead to frequent communication, heavy data transfer, and thus, increased energy consumption [114] [35]. Efficient synchronization mechanisms, such as event-driven architecture and differential synchronization can reduce the frequency and payload of communication.

Event-driven architecture uses events to trigger actions in different components. Instead of continuously polling for updates, components subscribe to relevant events and react when those events occur [115]. This saves resources by eliminating the unnecessary communication and operations [14].

Differential synchronization is a minimalistic synchronization technique where only the changes or differences between two versions of data are synchronized rather than transmitting the entire dataset [116]. This reduces the total amount of data that needs to be transferred.

However, sometimes polling cannot be avoided, such as with autosaving. In these situations, it is recommended to use the largest possible polling interval [34]. Similarly, it is more energy efficient to react to events as seldomly as possible. For example, input could be validated only when user moves on to the next input field, instead of reacting to every letter of the input.

6.7 Data format

Sending less data over the network can reduce energy consumption [14]. Larger file sizes typically require more storage space, more processing power, and more network bandwidth, which can result in higher energy consumption. Different file formats may support different features or data types, which can impact the file size.

6.7.1 Text-based data

In addition to bandwidth usage, text-based file formats involve extra computational cost from converting human-readable text data into a structured machine-readable format. This conversion consumes CPU and memory, which contributes to energy consumption. For text-based data, some of the most commonly used file formats include XML, JSON and CSV. These are compared in Table 7.

Table 7. File formats with examples of size overhead. [111] [117] [118] [119] [120]

	XML	JSON	CSV
Syntax	<price>3</price>	{ price: 3 }	;3
Payload	1	1	1
Overhead	15	11	1
Bandwidth	high	medium	low
CPU	high	medium	low
Complex data	yes	yes	no

As Table *7* shows CSV can be the most energy efficient option due to its low bandwidth and CPU usage. However, it does not support complex hierarchies of data, and therefore, JSON and XML are often more suitable choices. Out of these, JSON uses less bandwidth, CPU, and memory compared to XML, making it a more efficient option [117].

6.7.2 Images

The image format and compression algorithm can significantly affect the file size and energy consumption related to data transfer. Some of the most common image encoding/decoding algorithms are listed in Table *8* with a rough indication of the typical file size categories associated with each algorithm.

Table 8. Image formats [12] [11] [121] [122] [123] [124].

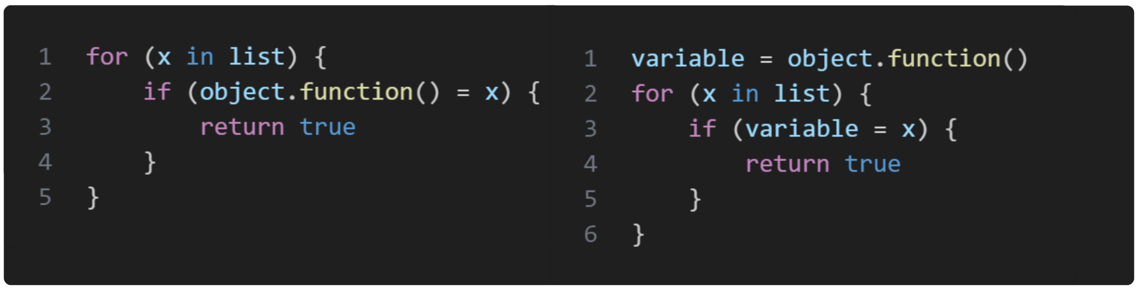
Format	Type	File size	Suitable for
SVG	vector	small	icons, logos, illustrations
WebP	lossy & lossless	small-medium	web images
GIF	lossless	small-medium	simple pictures and animations
JPEG	lossy	medium-large	images
PNG	lossless	large	high-quality images

More specific comparison of the energy efficiencies of these algorithms would require further research but based on the file size, SVG and WebP formats are the most efficient options [12]. However, the choice of image format and compression algorithm depends on various factors of the specific use case, including image content, desired image quality, and platform compatibility.

6.8 Code implementation

Code that consists of **fewer instructions** typically results in a more efficient program that consumes less computational resources, memory, and ultimately, less energy. By focusing on the code's logic and structure, energy efficiency can be optimized. This can be achieved by following good coding practices such as **avoiding global variables**, reducing the complexity inside loops, and **caching** the results of frequently performed calculations to variables [125].

Program *1* shows how complexity can be reduced in loops. Repeated function calls can be avoided by storing the result to a variable. While the program on the left appears to be simpler, and consists of fewer lines of code, the program on the right is actually more energy efficient.

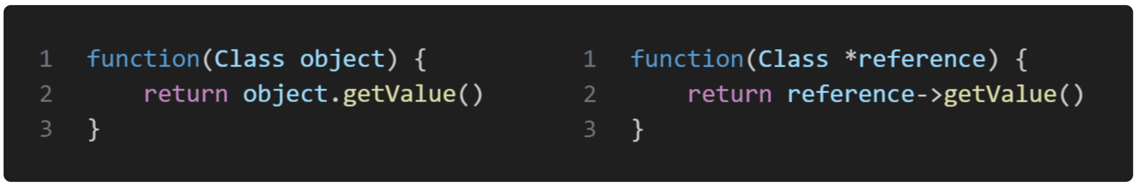


```
for (x in list) {
    if (object.function() = x) {
        return true
    }
}
```

```
variable = object.function()
for (x in list) {
    if (variable = x) {
        return true
    }
}
```

Program 1. Avoiding repeated function calls in a loop [125].

Some languages, such as C and C++, allow the programmer to choose the **parameter-passing strategy** manually, in which case energy can be saved by passing a reference instead of a copy of the object. This is shown in Program *2*, where the left side passes a copy of the object, and right side passes a reference. Higher-level languages like JavaScript and Python typically make this choice on the programmer's behalf.



```
function(Class object) {
    return object.getValue()
}
```

```
function(Class *reference) {
    return reference->getValue()
}
```

Program 2. Pass reference instead of a copy of the object [125].

Another aspect of code implementation is logging, which is used for recording and storing information about a program's execution, errors, and events to monitor the program’s correct behavior and simplify bug reporting. However, intensive logging can lead to frequent data transfer. **Avoiding unnecessary logging** can result in energy savings [126] [78].

It is also worth noting that more specific energy optimization techniques might depend on the programming language used. There can be found a lot of language specific resources in optimizing the energy consumption, such as [127] (Java) and [128] (C++). Additionally, **refactoring** can help keep the code simple and improve energy efficiency [3] [65] [129] [130].

7. USER EXPERIENCE

Ultimately, users use software to achieve a goal. This can be for instance, finding a piece of information, making a purchase, or submitting a form. The amount of energy consumed in the process depends on how the software is implemented and how efficiently the hardware is utilized. However, it is also influenced by the content presented to the user, and the time the user takes to achieve their goal. These can be optimized with sustainable user experience (UX) design.

This chapter describes how different aspects of UX design can be optimized for better energy efficiency. These include various usability factors and user interface (UI) design elements.

7.1 Usability

The longer a user uses a software, the more energy is consumed [14]. To improve the user's efficiency in achieving their goal, it is important to minimize wasted time, which includes activities such as navigating and searching for information, waiting for content to load, recovering from possible error situations, and learning how to use the software. [12] [11] [131]

Creating an intuitive and consistent user experience by using **familiar elements and features** in the design allows users to predict how elements behave based on past interactions. This can increase the learnability of the application and help the user achieve their task more efficiently [131].

Another way to increase the user’s efficiency is by **error prevention** and effective **error handling.** These can prevent users from making mistakes and help recover from error situations more effectively. This can be done for example, by validating user inputs and showing error messages that are informative, actionable, and easy to understand [131].

7.1.1 Content

The way content is organized and presented to a user can have a significant impact on how efficiently the user can achieve their goal. Various techniques can be used to make content perform well and minimize the time user spends on searching for information and navigating through the application.

To maximize efficiency, **information architecture** (IA) should be designed in a way that helps users find what they need as quickly as possible [12] [11]. This can be achieved, for example, by organizing content into logical categories and using clear and descriptive labels. Additionally, implementing a search functionality, and providing guidance to the user's desired content through effective navigation menus and breadcrumbs can further increase the user's efficiency [131].

Moreover, content should incite users to take action quickly [11]. This can be done, for example, by using action-oriented verbs or visual cues such as arrows or bold colors to highlight the action.

7.1.2 User Preferences

Users can be empowered to actively participate in energy efficiency efforts by incorporating user preferences into the software application. Allowing users to customize their preferences regarding energy consumption and energy critical features can help improve energy efficiency [78].

This can be achieved by informing users about energy intense operations and providing the option to customize, enable or disable certain features [78]. For example, users could be allowed to adjust data synchronization intervals or disable automatic background updates. Moreover, applications could feature a power saver (or eco) mode in which the user experience can drop in order to optimize energy consumption [78].

7.2 User Interface Design

The user interface (UI) can be designed to help increase the energy efficiency of a software application by implementing different optimization techniques. Different types of content and visual effects can be optimized to reduce data transfer and decrease loading times.

One effective way to optimize the UI is by adopting a minimalist approach. This can be achieved by eliminating unnecessary elements and rejecting ideas that cannot be justified. [12] Fewer visual elements require fewer system resources and less data that needs to be transferred.

7.2.1 Images

Images are one of the biggest contributors to the amount of data transferred on most websites [12]. It should be carefully assessed whether an image brings enough value to the page to justify the energy cost. If so, there are several options to reduce the file size of an image. These include reducing the amount of color variation in the image, and using more efficient file formats, like SVG, as discussed in section 6.7.2 [12].

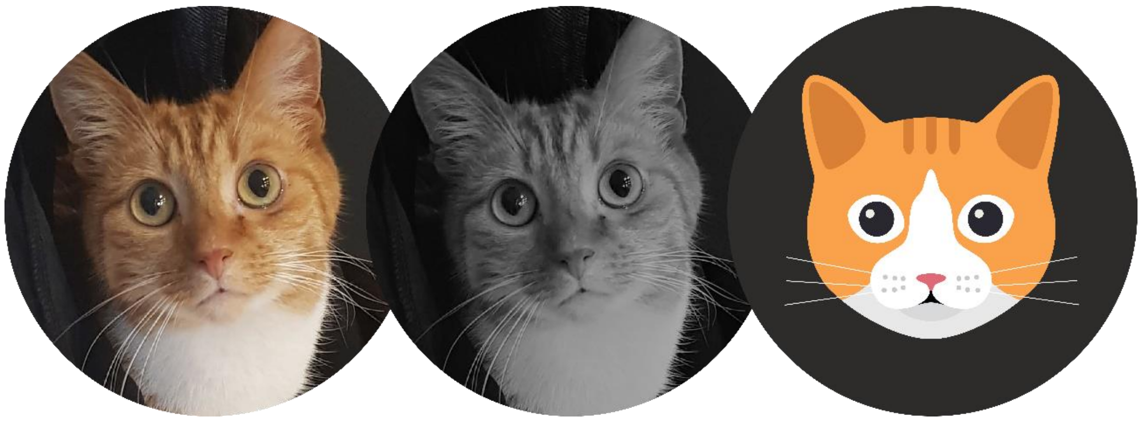


Figure 23. Images with reduced file size (1.17 Mt → 858 kt → 12 kt).

Another technique to reduce the file size is using lower resolution or blurring some parts of the image that are not as important, such as the background. By having different versions of the image with different resolutions, images can be loaded at the correct scale for each screen size [113]. Moreover, one possible alternative is to have images disabled by default, prompting users to enable image display or clicking to load an image [132].

7.2.2 Videos

Using videos can be an effective way to engage users and convey information [11]. However, videos are one of the largest contributors to the overall environmental impact of the internet. Therefore, videos should be used mindfully, and designs that involve automatically playing videos, like video backgrounds, should be avoided. Placing a play button in front of the video ensures that it only loads when the user explicitly chooses to watch it. This enhances the user experience and significantly reduces the unnecessary streaming of data [12].

Additionally, given that a single second of video content consumes more data compared to a full-screen JPEG image, videos should be kept short and impactful. This reduces energy consumption and saves users' time [12]. Moreover, the video file size can be minimized by reducing the resolution and framerate of the video [113].

7.2.3 Colors

Different colors of light have different wavelengths and energies. For example, red light has a longer wavelength and lower energy than blue light. When a screen displays a particular color, it is essentially emitting light of a specific wavelength and energy. The color temperature refers to the balance of blue and red light emitted by the screen. Cooler colors, which have a higher proportion of blue light, consume more energy than warmer colors, which have a higher proportion of red light. [133]

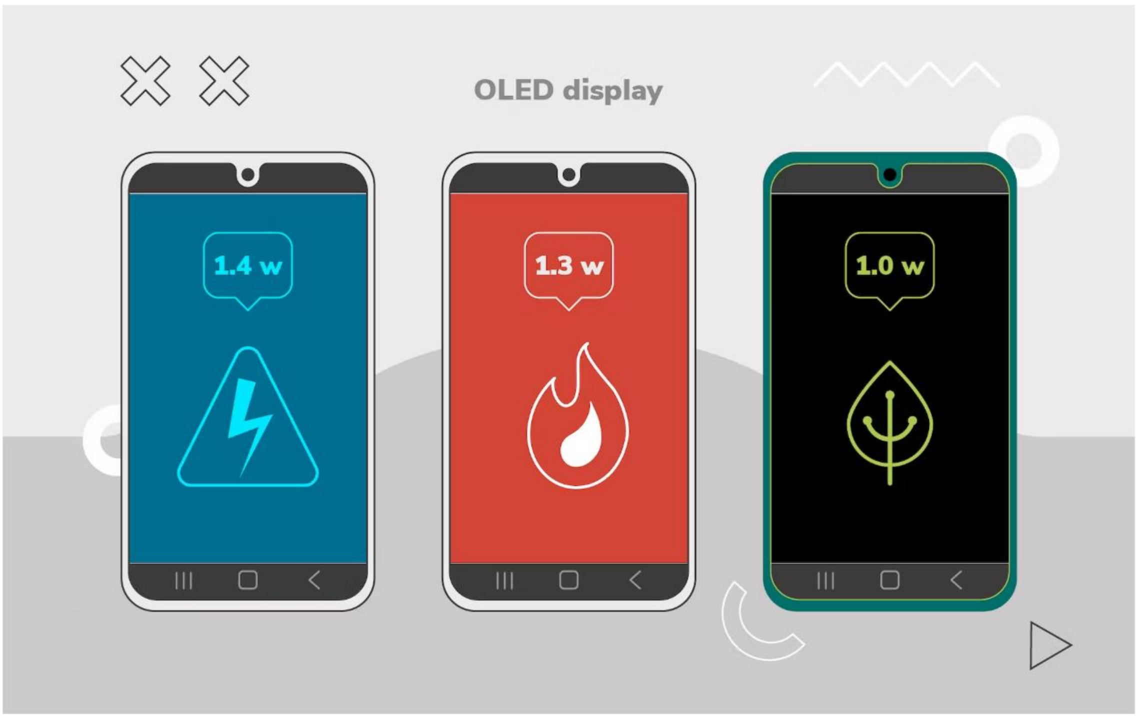


Figure 24. Colors on OLED display [134].

OLED (Organic Light-Emitting Diode) screens illuminate each pixel individually. This means that the pixels emit their own light, and the brightness of a pixel determines how much light it emits and therefore energy it consumes. When a pixel is displaying a dark color such as black, it is essentially turned off and does not emit any light, which saves a lot of energy. However, this does not apply to **LCD** (Liquid Crystal Display) screens, which have a permanent backlight, and use about the same amount of energy regardless of the color on the display [12].

7.2.4 Fonts

The availability of commercial and open-source web fonts has significantly increased the number of typographic options for user interfaces. However, their usage can result in a significant environmental impact due to the increased amount of data transfer and server requests required to load a webpage [12] [11].

The most energy efficient option is using system fonts, such as Arial, Times New Roman, Helvetica on Apple devices, and Roboto on Android, as they come pre-installed on devices. These fonts do not require additional server requests or data transfer. However, their use may limit creative freedom, and their appearance may vary across different devices, leading to a reduced level of control over presentation [12].

Having control over the font files allows for optimization. Various tools can be used to reduce the size of the font file and stripe out unused characters, such as special alphabets for certain languages [12]. However, it is worth noting that the licenses on some fonts may limit modifications.

8. CONCLUSION

The goal of this work was to explore different factors that impact the environmental sustainability of software applications and investigate how these factors can be optimized during the software development process. This work goes over a wide range of aspects within software development, and various optimization strategies were found on each area. The guidelines presented in this work include several approaches and actionable practices that can be incorporated into the development process. A summary of the guidelines and their possible trade-offs are listed in the appendices with references to section numbers in which they occur in this work.

It is worth noting that all the guidelines are not suitable or necessary for all software applications. Different software applications have varying computational demands, data processing requirements, and usage patterns. This influences the specific areas where optimizations can be applied, and which practices are realistic to implement. The strategies and practices for sustainability need to be tailored to the unique characteristics and requirements of each application.

Consistently measuring the energy consumption and emissions of the software can help give more detailed insight on which aspects and components are the most significant contributors to the energy consumption and carbon footprint. This may provide a better understanding of which optimization strategies could be the most effective for the specific software.

For further research, the impact of adapting these guidelines could be studied in different applications. This would help to get a broad idea of the effectiveness of the guidelines at different scales. Moreover, the perceived impact of the trade-offs related to the guidelines could also be measured.

REFERENCES

[1] GreenIT, "The environmental footprint of the digital world," 2019. Available: https://www.greenit.fr/wp-content/uploads/2019/11/GREENIT_EENM_etude_EN_accessible.pdf.

[2] The Shift Project, "Lean ICT: Towards Digital Sobriety," 2019. Available: https://theshiftproject.org/wp-content/uploads/2019/03/Lean-ICT-Report_The-Shift-Project_2019.pdf.

[3] L. Ardito, G. Procaccianti, M. Torchiano and A. Vetrò, "Understanding Green Software Development: A Conceptual Framework," *IT Professional,* vol. 17, no. 1, pp. 44-50, Jan.-Feb. 2015. Available: https://doi.org/10.1109/MITP.2015.16.

[4] M. Dastbaz, C. Pattinson and B. Akhgar, "Green Information Technology: A Sustainable Approach," Morgan Kaufmann Publishers Inc. 2015. Print. Available: https://doi.org/10.1016/C2014-0-00029-9.

[5] Intergovernmental Panel on Climate Change (IPCC), "Energy Systems," *Climate Change 2014: Mitigation of Climate Change,* pp. 511–598. 2015. Available: https://pure.iiasa.ac.at/id/eprint/11118/.

[6] Green Software Foundation, ”Carbon Awareness,” [Accessed Jun 20, 2023]. Available: https://learn.greensoftware.foundation/carbon-awareness.

[7] International Energy Agency (IEA), "Energy Technology Perspectives 2020," 2020. Available: https://www.iea.org/reports/energy-technology-perspectives-2020.

[8] H. Ritchie, M. Roser and P. Rosado, "Energy Mix," 2022. [Accessed Jul 13, 2023] Available: https://ourworldindata.org/energy-mix#.

[9] P. Wiesner, I. Behnke, D. Scheinert, K. Gontarska and L. Thamsen, "Let’s Wait Awhile: How Temporal Workload Shifting Can Reduce Carbon Emissions in the Cloud," *22nd International Middleware Conference (Middleware ’21),* ACM. 2021. Available: https://doi.org/10.1145/3464298.3493399.

[10] M. Á. Moraga and M. F. Bertoa, "Green Software Measurement," *Green in Software Engineering,* pp. 261-282. Springer International Publishing. 2015. Available: https://doi-org.libproxy.tuni.fi/10.1007/978-3-319-08581-4_11.

[11] T. Frick, "Designing for Sustainability : a Guide to Building Greener Digital Products and Services," O’Reilly. 2016. Print.

[12] T. Greenwood, Sustainable Web Design, A Book Apart, 2021. Print.

[13] E. Capra, C. Francalanci and S. A. Slaughter, "Is software “green”? Application development environments and energy efficiency in open source applications," *Information and Software Technology,* 54.1, pp. 60-71. 2012. Available: https://doi.org/10.1016/j.infsof.2011.07.005.

[14] L. Ardito and M. Morisio, "Green IT – Available data and guidelines for reducing energy consumption in IT systems," *Sustainable Computing: Informatics and Systems,* vol. 4, no. 1, pp. 24-32. 2014. Available: https://doi.org/10.1016/j.suscom.2013.09.001.

[15] F. Goekkus, "Energy Efficient Programming," University of Zurich. Dec 1, 2013. Available https://files.ifi.uzh.ch/hilty/t/examples/bachelor/Energy_Efficient_Programming_G%C3%B6kkus.pdf.

[16] T. Johann, M. Dick, S. Naumann and E. Kern, "How to measure energy-efficiency of software: Metrics and measurement results," *2012 First International Workshop on Green and Sustainable Software (GREENS),* pp. 51-54. 2012. Available: https://www.doi.org/10.1109/GREENS.2012.6224256.

[17] Green Software Foundation, "Measurement," [Accessed Jul 10, 2023]. Available: https://learn.greensoftware.foundation/measurement.

[18] Wholegrain Digital, "Website Carbon Calculator," [Accessed Jul 10, 2023]. Available: https://www.websitecarbon.com.

[19] Mightybytes, "Ecograder," [Accessed Jul 10, 2023]. Available: https://ecograder.com/.

[20] Greenspector, [Accessed Jul 10, 2023]. Available: https://greenspector.com/en/home/.

[21] University of Lille and Inria, "PowerAPI," [Accessed Jul 10, 2023]. Available: https://powerapi-ng.github.io/index.html.

[22] CAST, "CAST Highlight," [Accessed Jul 10, 2023]. Available: https://learn.castsoftware.com/green-software.

[23] Google, "Lighthouse," [Accessed Jul 10, 2023]. Available: https://developer.chrome.com/docs/lighthouse/.

[24] Microsoft, "Emissions Impact Dashboard," [Accessed Jul 10, 2023]. Available: https://www.microsoft.com/en-us/sustainability/emissions-impact-dashboard.

[25] Google, "Carbon Footprint," [Accessed Jul 10, 2023]. Available: https://cloud.google.com/carbon-footprint.

[26] Cloud Carbon Footprint, [Accessed Jul 10, 2023]. Available: https://www.cloudcarbonfootprint.org.

[27] Green Software Foundation, "Software Carbon Intensity (SCI) Specification," [Accessed Jul 10, 2023]. Available: https://github.com/Green-Software-Foundation/sci.

[28] Greenpeace, ”Giude to Greener Electronics,” 2017. Available: https://www.greenpeace.org/usa/reports/greener-electronics-2017/.

[29] Green Software Foundation, "Hardware Efficiency," [Accessed Jun 18, 2023] Available: https://learn.greensoftware.foundation/hardware-efficiency.

[30] L. Hilty and J. Bieser, "Opportunities and Risks of Digitalization for Climate Protection in Switzerland," University of Zurich. 2017. Available: https://doi.org/10.5167/uzh-141128.

[31] R. Verdecchia, P. Lago, C. Ebert and C. d. Vries, "Green IT and Green Software," *IEEE Software,* vol. 38, no. 6, pp. 7-15. 2021. Available: https://www.doi.org/10.1109/MS.2021.3102254.

[32] Green Software Foundation, "Energy Efficiency," [Accessed Jul 13, 2023]. Available: https://learn.greensoftware.foundation/energy-efficiency.

[33] L. A. Barroso and U. Hölzle, "The Case for Energy-Proportional Computing," *Computer,* vol. 40, no. 12, pp. 33-37, 2007. Available: https://doi.org/10.1109/MC.2007.443.

[34] P. Larsson, "Energy-Efficient Software Guidelines," Intel. 2011. Available: https://www.intel.com/content/dam/develop/external/us/en/documents/energy-efficient-software-guidelines-v3-4-10-11-140545.pdf.

[35] K. Eder ja J. P. Gallagher, "Energy-Aware Software Engineering," *ICT - Energy Concepts for Energy Efficiency and Sustainability,* InTech, Mar. 22, 2017. Available: https://doi.org/10.5772/65985.

[36] Google, "Google Apps: Energy Efficiency in the Cloud," 2012. Available: https://static.googleusercontent.com/media/www.google.com/en//green/pdf/google-apps.pdf.

[37] J.-H. Huh, "Server Operation and Virtualization to Save Energy and Cost in Future Sustainable Computing," *Sustainability,* vol. 10, no. 6, 2018. Available: https://doi.org/10.3390/su10061919.

[38] E. A. Santos, C. McLean, C. Solinas and A. Hindle, "How does docker affect energy consumption? Evaluating workloads in and out of Docker containers," *Journal of Systems and Software,* vol. 146, pp. 14-25. 2018. Available: https://doi.org/10.1016/j.jss.2018.07.077.

[39] Y. Jin, Y. Wen and Q. Chen, "Energy efficiency and server virtualization in data centers: An empirical investigation," *2012 Proceedings IEEE INFOCOM Workshops,* pp. 133-138. 2012. Available: https://www.doi.org/10.1109/INFCOMW.2012.6193474.

[40] O. helin, "A Study on Virtualization and Energy Efficiency Using Linux," 2012. Available: https://trepo.tuni.fi//handle/123456789/20918.

[41] R. Long, "Use of containerisation as an alternative to full virtualisation in grid environments," vol. 664, no. 2. 2015. Available: https://www.doi.org/10.1088/1742-6596/664/2/022028.

[42] R. Morabito, "Power Consumption of Virtualization Technologies: An Empirical Investigation," *2015 IEEE/ACM 8th International Conference on Utility and Cloud Computing (UCC),* pp. 522-527. 2015. Available: https://www.doi.org/10.1109/UCC.2015.93.

[43] Synopsys, "DevOps," [Accessed Jul 17, 2023]. Available: https://www.synopsys.com/glossary/what-is-devops.html.

[44] V. Avelar, D. Azevedo and A. French, "PUE: A Comprehensive Examination of the Metric," The Green Grid. 2014. Available: https://datacenters.lbl.gov/sites/default/files/WP49-PUE%20A%20Comprehensive%20Examination%20of%20the%20Metric_v6.pdf.

[45] Uptime Institute, "Uptime Institute Global Data Center Survey 2021," 2021. Available: https://uptimeinstitute.com/2021-data-center-industry-survey-results.

[46] Electricity Maps, [Accessed Nov 19, 2023]. Available: https://app.electricitymaps.com/map.

[47] K. Gallaba, "Improving the Robustness and Efficiency of Continuous Integration and Deployment," *IEEE International Conference on Software Maintenance and Evolution (ICSME),* pp. 619-623. 2019. Available: https://www.doi.org/10.1109/ICSME.2019.00099.

[48] M. Hilton, T. Tunnell, K. Huang, D. Marinov and D. Dig, "Usage, costs, and benefits of continuous integration in open-source projects," *31st IEEE/ACM International Conference on Automated Software Engineering (ASE),* pp. 426-437. 2016.

[49] R. Abdalkareem, S. Mujahid, E. Shihab and J. Rilling, "Which Commits Can Be CI Skipped?," *IEEE Transactions on Software Engineering,* vol. 47, no. 3, pp. 448-463. 2021. Available: https://www.doi.org/10.1109/TSE.2019.2897300.

[50] E. Kern, S. Naumann and M. Dick, "Processes for Green and Sustainable Software EngineeringProcesses for Green and Sustainable Software Engineering," *Green in Software Engineering,* Springer International Publishing. pp. 61-81. 2015. Available: https://doi.org/10.1007/978-3-319-08581-4_3.

[51] Adobe Communications Team, "Waterfall Methodology: A Complete Guide," *Adobe Experience Cloud Blog,* Mar 18, 2022. [Accessed Jul 19, 2023]. Available: https://business.adobe.com/blog/basics/waterfall.

[52] Software SOLVED, "Our Approach: Our Methodology," [Accessed Jul 19, 2023]. Available: https://www.softwaresolved.com/our-methodology.

[53] N. Rashid and S. U. Khan, "Agile Practices for Global Software Development Vendors in the Development of Green and Sustainable Software," *Journal of Software : Evolution and Process,* vol. 30, no. 10. 2018. Available: https://doi-org.libproxy.tuni.fi/10.1002/smr.1964.

[54] Emmit A. Scott, Jr, "SPA Design and Architecture: Understanding single-page web applications," Manning Publications. 2015. Print.

[55] T. F. Iskandar, M. Lubis, T. F. Kusumasari and A. R. Lubis, "Comparison Between Client-Side and Server-Side Rendering in the Web Development," *IOP Conference Series: Materials Science and Engineering,* vol. 801, no. 1. 2022. Available: https://doi.org/10.1088/1757-899X/801/1/012136.

[56] Vue.js, "Server-Side Rendering (SSR)," *Vue 3 documentation,* [Accessed Jul 26, 2023]. Available: https://vuejs.org/guide/scaling-up/ssr.html.

[57] Lindley, Cody, "DOM Enlightenment," Sebastopol: O'Reilly Media, Incorporated, 2013. Print.

[58] Green Software Foundation, "Avoid an excessive DOM size," *Green Software Patterns,* [Accessed Jul 6, 2023]. Available: https://patterns.greensoftware.foundation/catalog/web/avoid-excessive-dom-size/.

[59] R. Ollila, N. Mäkitalo and T. Mikkonen, "Modern Web Frameworks : A Comparison of Rendering Performance," *Journal of Web Engineering,* vol. 21 , no. 3 , pp. 789-813. 2022. Available: https://doi.org/10.13052/jwe1540-9589.21311.

[60] R. N. Diniz-Junior, C. C. L. Figueiredo, G. De S.Russo, M. R. G. Bahiense-Junior, M. V. Arbex, L. M. Dos Santos, R. F. Da Rocha, R. R. Bezerra and F. T. Giuntini, "Evaluating the performance of web rendering technologies based on JavaScript: Angular, React, and Vue," *XVLIII Latin American Computer Conference (CLEI),* pp. 1-9. 2022. Available: https://www.doi.org/10.1109/CLEI56649.2022.9959901.

[61] Stefan Krause, "js-framework-benchmark," [Accessed Jul 6, 2023]. Available: https://rawgit.com/krausest/js-framework-benchmark/master/webdriver-ts-results/table.html.

[62] J. De Macedo, R. Abreu, R. Pereira and J. Saraiva, "WebAssembly versus JavaScript: Energy and Runtime Performance," *International Conference on ICT for Sustainability (ICT4S),* pp. 24-34. 2022. Available: https://www.doi.org/10.1109/ICT4S55073.2022.00014.

[63] Mozilla, "WebAssembly," *mdn web docs,* Last modified on May 31, 2023. [Accessed Jul 6, 2023]. Available: https://developer.mozilla.org/en-US/docs/WebAssembly.

[64] M. v. Hasselt, K. Huijzendveld, N. Noort, S. d. Ruijter, T. Islam and I. Malavolta, "Comparing the Energy Efficiency of WebAssembly and JavaScript in Web Applications on Android Mobile Devices," *EASE '22: Proceedings of the 26th International Conference on Evaluation and Assessment in Software Engineering,* pp. 140–149. 2022. Available: https://doi.org/10.1145/3530019.3530034.

[65] G. Pinto, F. Soares-Neto and F. Castor, "Refactoring for Energy Efficiency: A Reflection on the State of the Art," *IEEE/ACM 4th International Workshop on Green and Sustainable Software,* pp. 29-35. 2015. Available: https://www.doi.org/10.1109/GREENS.2015.12.

[66] D. Li and W. G. J. Halfond, "An Investigation into Energy-Saving Programming Practices for Android Smartphone App Development," ACM. 2014. Available: https://www.doi.org/10.1145/2593743.2593750.

[67] A. King, "Parse, don’t validate," Nov 5, 2019. [Accessed Jul 6, 2023]. Available: https://lexi-lambda.github.io/blog/2019/11/05/parse-don-t-validate/.

[68] A. Bansal, "Performing Calculations in the Database vs. the Application," Oct 5, 2020. [Accessed Jul 11, 2023]. Available: https://www.baeldung.com/calculations-in-db-vs-app.

[69] Y. Z. AYIK and F. KAHVECİ, "COMPARISON OF CALCULATING OPERATIONS PERFORMANCES ON DATABASE SERVER AND WEB SERVER," *The journal of international social research,* vol. 12, no. 63, pp. 1324–1329. 2019. Available: https://doi.org/10.17719/jisr.2019.3321.

[70] R. Wang and Z. Yang, "SQL vs NoSQL: A Performance Comparison," [Accessed Jul 11, 2023]. Available: https://www.cs.rochester.edu/courses/261/fall2017/termpaper/submissions/06/Paper.pdf.

[71] Y. Li and S. Manoharan, "A performance comparison of SQL and NoSQL databases," *IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM),* pp. 15–19. 2013. Available: https://doi.org/10.1109/PACRIM.2013.6625441.

[72] TechEmpower, "Web Framework Benchmarks," Jul 19, 2022. [Accessed Jul 6, 2023]. Available: https://www.techempower.com/benchmarks/#section=data-r21.

[73] The Benchmarker, "Web Frameworks Benchmark," Last Update: Jul 2, 2023. [Accessed Jul 6, 2023]. Available: https://web-frameworks-benchmark.netlify.app/.

[74] A. Carroll, "Understanding and Reducing Smartphone Energy Consumption," 2017. Available: https://doi.org/10.26190/unsworks/19721.

[75] Y.-W. Kwon and E. Tilevich, "Reducing the Energy Consumption of Mobile Applications Behind the Scenes," *IEEE International Conference on Software Maintenance,* pp. 170-179. 2013. Available: https://www.doi.org/10.1109/ICSM.2013.28.

[76] L. Corral, A. B. Georgiev, A. Sillitti and G. Succi, "A study of energy-aware implementation techniques: Redistribution of computational jobs in mobile apps," *Sustainable Computing: Informatics and Systems,* vol. 7, pp. 11-23. 2015. Available: https://doi.org/10.1016/j.suscom.2014.11.005.

[77] Google, "Background Work Overview," *Developer guides,* Last updated 2023-01-04. [Accessed Jul 5, 2023]. Available: https://developer.android.com/guide/background.

[78] L. Cruz ja R. Abreu, ”Catalog of Energy Patterns for Mobile Applications,” *Empirical software engineering : an international journal,* vol. 24, no. 4, pp. 2209-2235. 2019. Available: https://www.doi.org/10.1007/s10664-019-09682-0.

[79] S. Huber, L. Demetz and M. Felderer, "PWA vs the Others: A Comparative Study on the UI Energy-Efficiency of Progressive Web Apps," *Web Engineering,* vol. 12706, pp. 464-479. Springer International Publishing. 2021. Available: https://doi.org/10.1007/978-3-030-74296-6_35.

[80] J. Manner, "Black software — the energy unsustainability of software systems in the 21st century," *Oxford Open Energy,* vol. 2. 2023. Available: https://doi.org/10.1093/ooenergy/oiac011.

[81] I. Manotas, C. Bird, R. Zhang, D. Shepherd, C. Jaspan, C. Sadowski, L. Pollock and J. Clause, "An Empirical Study of Practitioners' Perspectives on Green Software Engineering," *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE),* pp. 237-248. 2016. Available: https://www.doi.org/10.1145/2884781.2884810.

[82] M. C. Oussalah, R. Brohan and O. Moustafa, "Object Metrics for Green Software," *Journal of software,* pp. 285–305. 2021. Available: https://www.doi.org/10.17706/jsw.16.6.285-305.

[83] E. Jagroep, J. M. van der Werf, S. Brinkkemper, L. Blom and R. van Vliet, "Extending software architecture views with an energy consumption perspective: A case study on resource consumption of enterprise software," *Computing,* vol. 99, no. 6. 2017. Available: https://www.doi.org/10.1007/s00607-016-0502-0.

[84] S. Bhattacharya, K. Gopinath, K. Rajamani and M. Gupta, "Software Bloat and Wasted Joules: Is Modularity a Hurdle to Green Software?," *Computer,* vol. 44, no. 9, pp. 97-101. 2011. Available: https://www.doi.org/10.1109/MC.2011.293.

[85] E. Capra, C. Francalanci and S. A. Slaughter, "Is software “green”? Application development environments and energy efficiency in open source applications," *Information and Software Technology,* vol. 54, no 1, pp. 60-71. 2012. Availale: https://doi-org.libproxy.tuni.fi/10.1016/j.infsof.2011.07.005.

[86] M. R. Ghidersa, "Software Architecture for Web Developers," [First edition]. Packt Publishing. 2023.

[87] Gabriel, A. Sabino, L. Lima, V. Barbosa, C. Brito, P. Rego, Iure and F. A. Silva, "Energy Consumption in Microservices Architectures: A Systematic Literature Review," 2023. Available: https://www.doi.org/10.21203/rs.3.rs-3069141/v1.

[88] E. Ampomah, M. Ezekiel and G. Abilimi, "Qualitative Assessment of Compiled, Interpreted and Hybrid Programming Languages," *Communications on Applied Electronics,* vol. 7, no. 7, pp. 8–13. 2017. Available: https://doi.org/10.5120/cae2017652685.

[89] R. Pereira, M. Couto, F. Ribeiro, R. Rua, J. Cunha, J. P. Fernandes and J. Saraiva, "Energy Efficiency across Programming Languages: How Do Energy, Time, and Memory Relate?," *Proceedings of the 10th ACM SIGPLAN International Conference on Software Language Engineering,* pp. 256–267. 2017. Available: https://sites.google.com/view/energy-efficiency-languages/home?authuser=0.

[90] A. Koskela, "TypeScript fannkuch-redux implementation skews results in paper," *Energy Efficiency in Programming Languages,* May 25, 2022. [Accessed Jul 9, 2023]. Available: https://github.com/greensoftwarelab/Energy-Languages/issues/34.

[91] moaz_mokhtar, "Energy Efficiency across Programming Languages," Apr 26, 2021. [Accessed Jul 9, 2023]. Available: https://www.reddit.com/r/rust/comments/mytmyu/energy_efficiency_across_programming_languages/.

[92] Green Software Lab, "Energy Efficiency in Programming Languages," Last updated on Feb 27, 2022. [Accessed Jun 21, 2023]. Available: https://github.com/greensoftwarelab/Energy-Languages.

[93] PYPL, "PopularitY of Programming Language," Last updated Jul 2023. [Accessed Jul 22, 2023]. Available: https://pypl.github.io/PYPL.html.

[94] NumPy, Last updated Jun 17, 2023. [Accessed Jul 9, 2023]. Avaliable: https://numpy.org/.

[95] SciPy, Last updated Jun 28, 2023. [Accessed Jul 9, 2023]. Available: https://scipy.org/.

[96] pandas, [Accessed Jul 9, 2023]. Available: https://pandas.pydata.org/about/.

[97] V8, "Getting started with embedding V8," [Accessed Jul 9, 2023]. Available: https://v8.dev/docs/embed.

[98] Node.js, "C/C++ addons with Node-API," *Node.js v20.4.0 documentation,* [Accessed Jul 09, 2023]. Available: https://nodejs.org/api/n-api.html.

[99] G. Pinto, F. Castor and Y. D. Liu, "Understanding Energy Behaviors of Thread Management Constructs," *SIGPLAN notices,* vol. 49, no. 10, pp. 345-360. 2014. Available: https://www.doi.org/10.1145/2714064.2660235.

[100] B. Zhong, M. Feng and C.-H. Lung, "A Green Computing Based Architecture Comparison and Analysis," *IEEE/ACM Int'l Conference on Green Computing and Communications & Int'l Conference on Cyber, Physical and Social Computing,* pp. 386-391. 2010. Available: https://www.doi.org/10.1109/GreenCom-CPSCom.2010.110.

[101] S. Murugesan and G. R. Gangadharan, "Harnessing Green IT: Principles and Practices," John Wiley & Sons, Incorporated. 2012. Available: https://doi.org/10.1002/9781118305393.

[102] J. Michanan, R. Dewri and M. J. Rutherford, "GreenC5: An adaptive, energy-aware collection for green software development," *Sustainable Computing: Informatics and Systems,* vol. 13, pp. 42-60, 2017. Available: https://doi.org/10.1016/j.suscom.2016.11.004.

[103] S. Bae, "JavaScript Data Structures and Algorithms: An Introduction to Understanding and Implementing Core Data Structure and Algorithm Fundamentals," Berkeley, CA: Apress L. P, 2019. Print. Available: https://doi.org/10.1007/978-1-4842-3988-9.

[104] Educative, "Big O Notation: A primer for beginning devs," Dec 26, 2019. [Accessed Jun 20, 2023] Available: https://www.educative.io/blog/a-big-o-primer-for-beginning-devs.

[105] "Big-O Cheat Sheet," [Accessed Jun 20, 2023]. Available: https://www.bigocheatsheet.com/.

[106] Mozilla, "Map," *JavaScript,* Last modified: Jun 20, 2023. [Accessed Jun 21, 2023]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Map.

[107] Python, "Built-in Types," *3.12.0 Documentation,* Last modified Nov 26, 2023. [Accessed Nov 27, 2023]. Available: https://docs.python.org/3/library/stdtypes.html#typesmapping.

[108] cplusplus.com, "C++ reference - std::map," [Accessed: Jun 21, 2023]. Available: https://cplusplus.com/reference/map/map/.

[109] cppreference.com, "std::map," *C++ reference,* Last modified 29 April 2023. [Accessed Jun 21, 2023]. Available: https://en.cppreference.com/w/cpp/container/map.

[110] A. Noureddine, A. Bourdon, R. Rouvoy and L. Seinturier, "A preliminary study of the impact of software engineering on GreenIT," *First International Workshop on Green and Sustainable Software (GREENS),* pp. 21-27. 2012. Available: https://www.doi.org/10.1109/GREENS.2012.6224251.

[111] M. Koljonen, ”Vihreä koodi – mitä on Green Coding?,” Apr 19, 2022. [Accessed Jul 7, 2023]. Available: https://www.cgi.com/fi/fi/blogi/mita-green-coding-on.

[112] Python, "functools — Higher-order functions and operations on callable objects," *3.11.4 Documentation,* Last updated Jul 24, 2023. [Accessed Jul 24, 2023]. Available: https://docs.python.org/3/library/functools.html.

[113] M. Dick, S. Naumann and A. Held, "Green Web Engineering: A Set of Principles to Support the Development and Operation of ‘Green’ Websites and Their Utilization During a Website’s Life Cycle," *WEBIST 2010 - Proceedings of the 6th International Conference on Web Information Systems and Technology,* 2010. Available: https://www.scitepress.org/papers/2010/28052/28052.pdf.

[114] G. Fagas, L. Gammaitoni, J. P. Gallagher and D. J. Paul, "ICT - Energy Concepts for Energy Efficiency and Sustainability," IntechOpen. 2017. Available: https://doi.org/10.5772/62522.

[115] AWS, "What is an Event-Driven Architecture?," [Accessed Jul 7, 2023]. Available: https://aws.amazon.com/event-driven-architecture/.

[116] N. Fraser, "Differential Synchronization," ACM. 2009. Available: https://static.googleusercontent.com/media/research.google.com/fi//pubs/archive/35605.pdf.

[117] R. Rianto, M. A. Rifansyah, R. Gunawan, I. Darmawan and A. Rahmatulloh, "Comparison of JSON and XML Data Formats in Document Stored NoSql Database Replication Processes," *International Journal on Advanced Science, Engineering and Information Technology,* vol. 11, no. 3, pp. 1150–56. 2021. Available: https://doi.org/10.18517/ijaseit.11.3.11570..

[118] S. Patni, "Introduction: XML and JSON," *Pro RESTful APIs with Micronaut,* Apress L. P. 2023. Available: https://doi.org/10.1007/978-1-4842-9200-6_3.

[119] Barthelemy NGOM, "Storage size and generation time in popular file formats," Mar 22, 2021. [Accessed Jul 24, 2023]. Available: https://www.adaltas.com/en/2021/03/22/performance-comparison-of-file-formats/.

[120] Aida NGOM, "Comparison of different file formats in Big Data," Jul 23, 2020. [Accessed Jul 24, 2023]. Available: https://www.adaltas.com/en/2020/07/23/benchmark-study-of-different-file-format/.

[121] E. Öztürk and A. Mesut, "Performance Evaluation of JPEG Standards, WebP and PNG in Terms of Compression Ratio and Time for Lossless Encoding," *2021 6th International Conference on Computer Science and Engineering (UBMK),* pp. 15-20. 2021. Available: https://www.doi.org/10.1109/UBMK52708.2021.9558922.

[122] M. Rashid, L. Ardito and M. Torchiano, "Energy Consumption Analysis of Image Encoding and Decoding Algorithms," *2015 IEEE/ACM 4th International Workshop on Green and Sustainable Software,* pp. 15-21. 2015. Available: https://www.doi.org/10.1109/GREENS.2015.10.

[123] E. Paiz-Reyes, N. Nunes-de-Lima and S. Yildirim-Yayilgan, "GIF Image Retrieval in Cloud Computing Environment," 2018. Available: https://doi.org/10.1007/978-3-319-93000-8_30.

[124] Google, "WebP Compression Study," Last updated Jun 29, 2023. [Accessed Jul 23, 2023]. Available: https://developers.google.com/speed/webp/docs/webp_study.

[125] J. Corral-García, F. Lemus-Prieto, J.-L. González-Sánchez and M.-Á. Pérez-Toledano, "Analysis of Energy Consumption and Optimization Techniques for Writing Energy-Efficient Code," *Electronics,* vol. 8, no. 10, pp. 1192. 2019. Available: https://www.doi.org/10.3390/electronics8101192.

[126] S. Chowdhury, S. D. Nardo, A. Hindle and Z. M. (. Jiang, "An exploratory study on assessing the energy impact of logging on Android applications," *Empirical Software Engineering,* vol. 23, no. 3, pp. 1422–1456. 2018. Available: https://doi-org.libproxy.tuni.fi/10.1007/s10664-017-9545-x.

[127] H. S. Zhu, C. Lin and Y. D. Liu, "A Programming Model for Sustainable Software," *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering,* vol. 1, pp. 767-777. 2015. Available: https://www.doi.org/10.1109/ICSE.2015.89.

[128] R. Grimes and M. Bancila, "Modern C++: Efficient and Scalable Application Development," 1st edition, Packt Publishing, 2018..

[129] C. Sahin, L. Pollock and J. Clause, "How do code refactorings affect energy usage?," *International Symposium on Empirical Software Engineering and Measurement,* pp. 1–10. ACM. 2014. Available: https://doi.org/10.1145/2652524.2652538.

[130] R. Sehgal, D. Mehrotra, R. Nagpal and R. Sharma, "Green software: Refactoring approach," *Journal of King Saud University. Computer and Information Sciences,* vol. 34, no. 7, pp. 4635–4643. 2022. Available: https://doi.org/10.1016/j.jksuci.2020.10.022.

[131] Voil, Nick de, "User Experience Foundations," BCS Learning & Development Limited. 2019.

[132] ClimateAction.tech, "Create Low-Carbon Images," [Accessed Jun 22, 2023]. Available: https://climateaction.tech/actions/create-low-carbon-images/.

[133] X. Chen, Y. Chen, Z. Ma and F. C. A. Fernandes, "How is Energy Consumed in Smartphone Display Applications?," ACM. 2013. Available: https://doi.org/10.1145/2444776.2444781.

[134] D. Rousset, "Creating Green Energy Efficient Progressive Web Apps," August 12th, 2021. [Accessed Jun 28, 2023]. Available: https://greensoftware.foundation/articles/creating-green-energy-efficient-progressive-web-apps.

APPENDIX A: ENERGY RELATED GUIDELINES FOR ENVIRONMENTALLY SUSTAINABLE SOFTWARE APPLICATIONS

Subject	Guidelines	Trade-Offs	Sec.
Carbon intensity	Move computation to a place where carbon intensity is lower, and do more computation when the circumstances are favorable and more renewable energy is available	Requires additional monitoring of carbon intensity and weather conditions	2.1
Measuring	Consistently measure the energy consumption and emissions of the software to monitor improvements and identify resource-intensive components	Can be time-consuming, requires tools and additional work	2.3

APPENDIX B: HARDWARE RELATED GUIDELINES FOR ENVIRONMENTALLY SUSTAINABLE SOFTWARE APPLICATIONS

Subject	Guidelines	Trade-Offs	Sec.
Extend hardware lifespan	Design software that can be run on older hardware	Can limit new features and increase resources required for testing	3.2
Increase hardware utilization	Move computation to public cloud, and employ virtualization and containerization technologies	Requires DevOps knowledge	3.4

APPENDIX C: DEVOPS & PROJECT MANAGEMENT RELATED GUIDELINES FOR ENVIRONMENTALLY SUSTAINABLE SOFTWARE APPLICATIONS

Subject	Guidelines	Trade-Offs	Sec.
Cloud	Choose a data center that is close to users, runs with renewable energy and has a low PUE	Requires further investigation of data centers, can be more expensive	4.1
CI/CD	Reduce the amount of automatization and workload in CI/CD pipelines	Slightly increased chance of unnoticed bugs	4.2
Methods	Use Agile methods, instead of waterfall, to reduce wasted efforts		4.3

APPENDIX D: WEB APPLICATION RELATED GUIDELINES FOR ENERGY EFFICIENCY

Subject	Guidelines	Trade-Offs	Sec.
Rendering	Use static site generation and consider using server-side rendering to reduce the total workload	Not suitable for dynamic content	5.1
Frameworks	Use lightweight frameworks, such as Vue or Preact for front-end, and Fastify or FastAPI for back-end	Might not support as many features, smaller community might not offer as much support, can contain more bugs	5.1.2 5.2.3
WebAssemby	Consider using WebAssembly for energy intensive operations	Requires advanced knowledge	5.1.3
HTTP communication	Consider bundling small HTTP requests into fewer, larger requests	Requires more specific routes, which can reduce reusability	5.2.1
HTTP parsing	Consider parsing the requests straight away rather than first validating, then parsing		5.2.1

Subject	Guidelines	Trade-Offs	Sec.
Database	Use efficient database queries by querying only the required data, and perform complex calculations and data operations in the application code		5.2.2
Mobile	Bundle background activities together, use Wi-Fi over cellular		5.3
Mobile platforms	Develop applications natively or with more efficient cross-platform frameworks, such as Flutter	Can require additional work and knowledge	5.3

APPENDIX E: SOFTWARE IMPLEMENTATION GUIDELINES FOR ENERGY EFFICIENCY

Subject	Guidelines	Trade-Offs	Sec.
Architecture	Implement proper object-oriented paradigm for modular, scalable, and reusable software	Requires advanced experience to implement properly	6.1
Software bloat	Avoid software bloat by being selective about external library usage	Might require more manual implementation	6.1
Programming language	Use efficient programming languages	Reduced readability and learnability, can be more time-consuming to develop	6.2
Concurrency	Consider multithreading when a large workload can be spread evenly		6.3
Data structures	Find the most efficient data structure for each use case		6.4
Algorithms	Focus on improving the algorithm’s mathematical complexity over the code implementation	Requires mathematical proficiency and increased work	6.4

Subject	Guidelines	Trade-Offs	Sec.
Cache	Use various caching mechanisms at code-level and with communication	Increased memory and programmer’s responsibility	6.5
Synchronization	Use efficient synchronization techniques and increase polling interval where it can’t be avoided		6.6
Data format	Use efficient data format that reduces the size of the file	Might not support as many features	6.7
Code implementation	Implement code with fewer instructions and adopt good coding practices		6.8

APPENDIX F: USER EXPERIENCE GUIDELINES FOR ENERGY EFFICIENCY

Subject	Guidelines	Trade-Offs	Sec.
Usability	Increase user's efficiency by using familiar features and elements, preventing errors, and handling error situations effectively	Limiting creativity in design	7.1
Content	Organize content logically and encourage users to take action quickly by highlighting important elements		7.1.1
User preferences	Allow energy intensive features to be customized or disabled		7.1.2
User interface	Design user interface with minimalistic approach	Limiting creativity in design	7.2
Images	Use correct resolution for each screen size	Requires multiple copies of the image	7.2.1
Image optimization	Reduce image file size by reducing color variation, blurring background and selecting efficient image formats	Limiting creativity in design, requires additional operations for optimization	7.2.1 6.7.2
Videos	Minimize videos and avoid auto-playing videos by placing a play button, minimize the resolution and framerate of the video	Limiting creativity in design, requires additional operations for optimization	7.2.2

Subject	Guidelines	Trade-Offs	Sec.
Colors	Use darker and warmer colors	Limiting creativity in design	7.2.3
Fonts	Use system fonts, and strip out unused characters from non-system font files	Limiting creativity in design and requires additional operations for optimization	7.2.4