Lessons learned from developing green software

Cruz, Luís; Heck, Petra

DOI
10.3920/9789004730779_012

Publication date
2025

Document Version
Final published version

Published in
Moral Design and Green Technology

CHAPTER 11

Lessons learned from developing green software

Luís Cruz and Petra Heck

Abstract

Technology brings exciting opportunities to improve our interactions with the natural surroundings. However, that same technological development might also negatively impact the environment. Every new technology has a carbon footprint, whether from its construction or operation. And most technological developments require software systems, and more recently AI-based software systems. For these software systems to positively impact our environment, they need to be developed and operated with sustainability in mind, also called 'green' in the discipline of software engineering.

This chapter explores various dimensions of sustainability in software system development, drawing on existing software quality frameworks. We highlight green software best practices for development and knowledge transfer. We examine AI-based software systems, emphasising the importance of energy efficiency and carbon impact in the next generation of intelligent systems. This entails considering decisions at different stages of the AI lifecycle, ranging from underlying design choices in training pipelines to selecting optimal hardware for training and serving models.

This chapter presents the intersection of green software, sustainable software engineering, and green AI as of major importance for future innovation. By prioritising sustainability in software development and AI, we can foster a more sustainable and eco-friendly future, with the potential to reduce energy consumption and mitigate the environmental impact of technology.

11.1 Innovation and environmental responsibility

In an era defined by technological marvels, our world stands at a crossroads where innovation and environmental responsibility converge. Software technology promises to transform how we interact with the natural world, offering the prospect of a sustainable future. For example, it is used to track climate patterns, biodiversity, water and air quality; it is also used in waste management systems to optimise waste collection routes, manage landfill operations, and so on. Yet, software takes its own toll in terms of impact in the environment. For example, running software worldwide requires massive amounts of electricity. It is estimated that, by 2030, it will be responsible for 13% of electricity consumption globally (EU, 2022). Hence,

we are faced with a formidable challenge: how can we harness the power of software without leaving a devastating ecological footprint in our wake?

To bend the curve of biodiversity loss, our attitude towards nature has to change fundamentally. This behavioural change can for instance be triggered by engaging smartphone apps that can recognise flora and fauna at species level. This helps raise awareness and create knowledge of our natural environment.

CASE 11.1

AI-enabled mobile application

Consider, a mobile application that is designed to store and analyse pictures of flora and fauna. It leverages an artificial intelligence (AI) model to identify species automatically and catalogue your pictures according to contemporary taxonomic tables. It includes both local storage, in the phone, and online storage, in a cloud server, so that you do not lose your records even if something happens to your phone.

One key feature of this app is uploading the pictures to a server in the cloud. There are several ways of implementing this feature. For example, one way could be that, as soon as a picture is taken, the app takes care of immediately uploading it. This way, the pictures are immediately backed up as soon as they are taken. Most users would appreciate having their pictures safely secured as soon as they are taken.

Imagine, however, that many of the app users enjoy going for a hike in the woods, where there is poor internet connectivity. While enjoying the hike, users also enjoy taking pictures of the interesting things they see along the way: fauna, flora, landscape, etc. However, that implies that the phone makes a new data connection with the server every time a picture is taken. With poor connectivity, it would probably mean that the phone would take such a long time to upload each picture that it would not be fast enough for all the pictures that the user is taking. To make things worse, the phone's battery level is getting low, but the app is eagerly trying to get the pictures uploaded and will not stop until the phone dies.

The scenario in CASE 11.1 motivates developers to find strategies to make sure their software is designed in the most energy efficient way. It challenges them to include strategies to test and monitor the energy efficiency of their code. On top of that, smartphones are now running AI features locally, without an internet connection. This means that we save energy by not using an internet connection, but we drain our battery to run these powerful AI models. The proliferation of large language models in 2021 takes this challenge to another level, yielding ever-growing models with billions of parameters. As we delve deeper into this chapter, we show that a

single training iteration of these AI models leads to a massive carbon footprint. This motivates a need for new standards and practices that make energy efficiency a *de facto* requirement for modern software.

In this chapter, we cover the various dimensions of sustainable software and AI engineering, providing insights into how these pressing issues are being tackled. We start by defining the concepts around software sustainability. Then we analyse the well-established software quality standard ISO 25000 and look at how it relates to the practical realms of developing green software and green AI. Along the way, we unveil best practices that not only guarantee the energy efficiency but also the collection of reliable energy measurements from software.

11.2 Sustainable software engineering

sustainable software engineering

Although sustainability is a widely used term, the definition of sustainable software engineering is not always clear. Within this chapter, we define it as 'the discipline that studies the process of creating software systems that create value in the long term without hindering its surroundings'. Despite being short, this definition gives us a starting point from which to identify software systems that are not sustainable. For example, if a software system is carbon efficient and eco-friendly but does not create value, it is not sustainable. Moreover, if a software system is eco-friendly but was designed in a way that makes it difficult to maintain (e.g. fixing security vulnerabilities), it is also not sustainable. The examples are numerous.

Software sustainability can be divided into five major dimensions (Becker *et al.*, 2015), as illustrated in Figure 11.1: environmental, social, individual, technical, and economical. We explain these five perspectives of software sustainability below.

environmental sustainability

Environmental sustainability relates to the long-term effects of software on natural systems. This dimension includes ecosystems, raw resources, climate change, water, pollution, waste, etc. This can be attributed for example to the power consumption of the infrastructure used to run the software (e.g., electricity used by data centres), the water consumption used to cool down supercomputers, the disposal of IT hardware to acquire new state-of-the-art replacements, and so on. The branch of software engineering that studies this dimension is known as green software.

social sustainability

Social sustainability is concerned with societal communities (groups of people, organisations) and the factors that erode trust in society. This dimension includes social equity, justice, employment, democracy, public health, public well-being, and so on. There is no doubt that software systems have a tremendous impact in our society and on individual people nowadays. This impact is often positive, but it can also have some negative consequences. A recurrent example of lack of social sustainability lies in many modern social media platforms, where we have witnessed the exploitation of their feed algorithms to disseminate fake news and mislead public opinion.

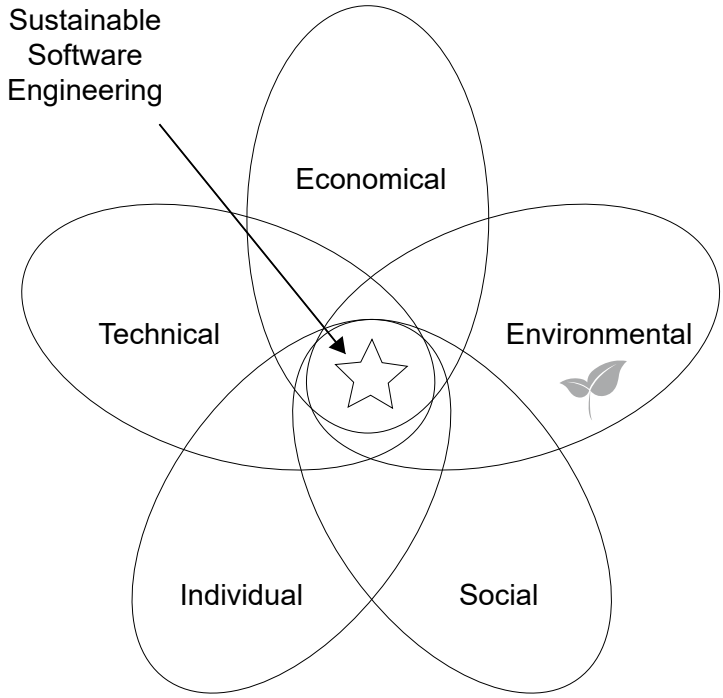

FIGURE 11.1 The five dimensions of sustainable software engineering

Individual sustainability refers to the well-being of the people involved in the development of the software. This includes mental and physical well-being, education, self-respect, skills, mobility, and so on. Several factors such as work-life balance, social environment, job autonomy, and physical health affect the well-being of tech professionals. Most of the work in this dimension aims at making sure that all the collaborators in a software team have a motivating, safe, and healthy working environment that fosters their contributions to the software project. Examples include promoting diversity within teams, sponsoring gym membership, defining clear career paths, organising team building activities, and so on.

individual sustainability

Technical sustainability refers to the longevity of information, systems, and infrastructure and their adequate evolution with changing surrounding conditions. It includes concerns such as maintenance, innovation, data integrity, etc. This is a dimension that has been widely addressed within software engineering communities. The lack of technical sustainability means, for example, that a system is not scalable and thus does not follow the growth of customers. It can also mean that whenever a new feature needs to be implemented, it is very complicated to do it in a way that will not break other features.

technical sustainability

Economic sustainability focuses on assets, capital and added value. It includes wealth creation, prosperity, profitability, capital investment, income, and so on.

economic sustainability

This is an essential dimension for any organisation. The operation of a software system must not lead its investors into bankruptcy. Even when we talk about toy software systems, if their contributors cannot support the costs of having a system available to its users, it will soon have to be decommissioned.

11.2.1 *Green software*

All these dimensions are important and interact with each other. For example, environmental sustainability is important for social sustainability and should not harm the economic sustainability of a project. However, some of these five dimensions have become more important throughout the history of software engineering. It is not difficult to convince a software organisation that they need to worry about the technical and economic sustainability of their software. For example, tech organisations take great pains to ensure that their business model is economically sustainable. Start-ups go through different rounds of funding where economical sustainability is the main concern. For example, if a fintech company decides to become more environmentally sustainable and moves all its software systems to data centres that run on clean energy, these data centres will probably be more expensive. Hence, the company will have to come up with a business model that is able to accommodate these extra costs, perhaps by charging their customers slightly higher fees. If customers do not deem these extra costs to be reasonable, this company might face serious economic issues – or simply shut down the idea at the first opportunity. Technical sustainability is also quite popular and is second only to economic sustainability. In the early versions of a software product, prototypes and proof-of-concepts are the main artefact being used to assess whether the product might be (economically) sustainable. Right after this stage, stakeholders start thinking about technical sustainability. In other words, software sustainability is a systemic concept and one dimension cannot be studied without considering the others.

green software

With that in mind, a new discipline is emerging in software engineering: green software. Green software covers activities across the whole software development lifecycle: planning, analysis, design, implementation, and maintenance. This includes, for example, making sure that the software is energy efficient, i.e. that it uses the least power – which is easier said than done (Cruz, 2021). With the emergence of this field, any software organisation or developer that wants to deliver high-quality software cannot succeed without considering Green Software practices.

11.2.2 *A quality perspective on green software*

ISO 25000

There have long been standards (since 1991!) that define what high quality means. ISO 9126 and its successor ISO 25000 (Systems and software Quality Requirements and Evaluation: SQuaRE) define quality characteristics for software systems,

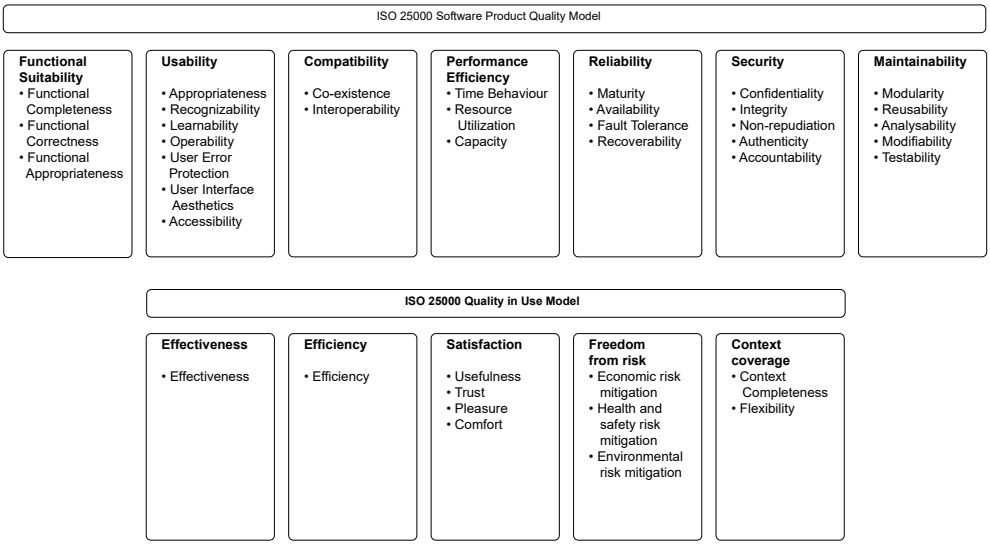

FIGURE 11.2 ISO 25000 quality model

including a measurement model with metrics. But it is only much more recently that the debate about green software has begun. So, one might think that green is a new quality characteristic that needs to be added to the ISO 25000 standards. However, as seen in Figure 11.2, which shows the quality characteristics for the inherent product quality and the emerging product quality (quality in use), ISO 25000 already includes the quality characteristics that define green software under the category '*Performance Efficiency*'. In fact, ISO 25000 goes one step further and explicitly states under the category '*Freedom from risk*' that the software system shall also mitigate any other environmental risk than just consuming unnecessary energy or storage.

So, in the light of ISO 25000, green software can be translated to software that scores high on the following two quality characteristics:

1. Performance efficiency: 'performance relative to the amount of resources used under stated conditions'. — performance efficiency
    a. time behaviour: 'degree to which the response and processing times and throughput rates of a product or system, when performing its functions, meet requirements'.
    b. resource utilisation: 'degree to which the amounts and types of resources used by a product or system, when performing its functions, meet requirements'.
    c. capacity: 'degree to which the maximum limits of a product or system parameter meet requirements (note: parameters can include the number of items that can be stored, the number of concurrent users, the

communication bandwidth, throughput of transactions, and size of database)'.

environmental risk mitigation

2. Environmental risk mitigation: 'degree to which a product or system mitigates the potential risk to property or the environment in the intended contexts of use'.

One could argue that when developing performance-efficient software, environmental risk is also mitigated, but of course more needs to be done to fully mitigate any harm to the environment. In the remainder of this chapter, we will focus on the first characteristic, related to the energy consumption of software, because this is what green software has been mostly focusing on. Energy consumption is seen as the biggest environmental risk for software development, especially with the advent of AI-based software systems that require huge amounts of data storage and computing power.

11.3 Measuring energy consumption

There is no perfect way to measure energy consumption. It mostly boils down to a trade-off between simplicity and accuracy. In other words, the easiest approach tends to be less accurate, and the most accurate approach tends to be difficult to set up.

Let us start with the easiest approach. To estimate energy consumption, one can simply measure the time a software takes to execute a given task. If task A takes more time to run than task B, one can assume that A takes more energy than B. This assumption works well under well-defined conditions where we have observed this to be the case. For example, if we are comparing single-threaded Central Processing Unit (CPU) intensive algorithms, the algorithm that takes more time will be asking the CPU to execute more work. The more work from the CPU, the more energy is spent.

This assumption is challenged when the software uses a large set of resources with dynamic configurations that we cannot entirely predict beforehand: for example, a mobile application can use different sensors (GPS, accelerometer, camera, touch, etc.), different CPU modes (low-power or high-power units), different actuators (screen with different brightness colours, speakers, haptics, etc.). Modelling these different interactions is far from straightforward. Moreover, it is very unlikely that these interactions are identical across two different executions of the same task or use case.

As an alternative, the most accurate solution is to collect exact power measurements during the execution of the software – i.e. using a hardware-based approach. Using power monitoring tools, such as the Monsoon Power Monitor presented in Figure 11.3, it is possible to collect power data from the different hardware components and log power measurements per instant of time.

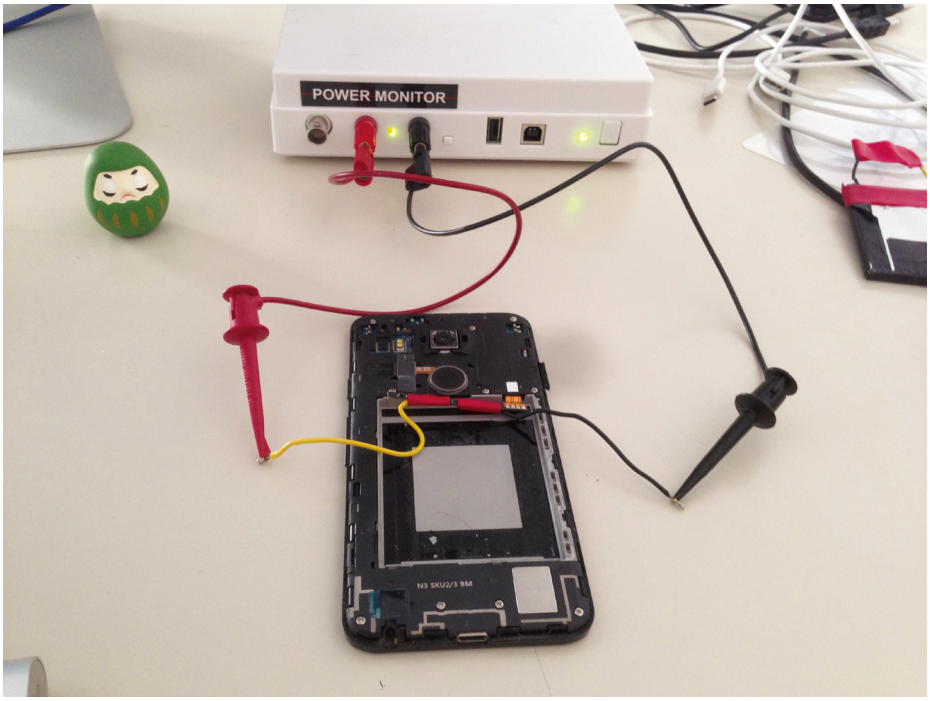

FIGURE 11.3 Energy monitoring setup for mobile app development

Unfortunately, this is far from easy. The figure above shows the power monitor connected to a smartphone. This setup required disassembling a smartphone and extracting its battery. That is suboptimal: mobile developers cannot be expected to recreate such a setup for the sake of improving energy efficiency.

An alternative is the use of software-based estimators of energy consumption. These estimators rely on indicators of the usage of different hardware resources (memory, CPU, and so on) to compute a value of the power consumption of a given software. A popular example is Intel® RAPL, which has been widely used in this area by researchers and tool developers. Depending on the scenario, these different alternatives need to be considered.

There are of course more challenges that need to be addressed before being able to measure software energy consumption. Software seldom runs in isolation: there is an operative system, other software applications, background processes, etc. that are running in parallel and contribute to the measured energy consumption.

To add to that, there are a multitude of other factors that also contribute to energy consumption: the power state of the different components (e.g., is the GPS already running or does it still need to be powered up?), their temperature, the temperature of the surroundings, and so on.

This means that if we measure the same software task twice, it is very unlikely that we will measure the same energy consumption.

There are several steps that need to be followed to collect reliable energy data. Below we have an extensive but not complete list of actions that should be taken before collecting energy data:

zen mode
- The first thing we need to ensure is that the only thing running in our system is the software we want to measure. Unfortunately, this is impossible in practice – our system will always have other tasks and things that it will run at the same time. Still, we must at least minimise all these competing tasks: all applications should be closed, and notifications should be turned off; only the required hardware should be connected (avoid USB drives, external disks, external displays, etc.); remove any unnecessary services running in the background (e.g., web server, file sharing, etc.); if you do not need an internet or intranet connection, switch off your network; cable is preferable to wireless – the energy consumption from a cable connection is more stable than from a wireless connection.

freeze your settings
- It is not possible to shut off the unnecessary things that run in our system. Still, we need to at least make sure that they will behave the same across all sets of measurements. Thus, we must fix and report some configuration settings. One good example is the brightness and resolution of your screen – report the exact value and make sure it stays the same throughout the measurement. A common mistake is to keep the automatic brightness adjustment on – this leads to major errors when measuring energy efficiency in mobile apps.

warm up
- Run dummy tasks a few times before keeping track of the measurements. Energy consumption is greatly affected by the temperature of your hardware. The higher the temperature, the higher the resistance of electrical conductors, leading to higher dissipation and consequently more energy consumption. If we start measurements right after the workstation comes back from sleep mode, it will probably be relatively cool. As soon as we start executing our energy tests, the temperature of the computer will rise until it reaches a plateau. This means that the first measurements will show less energy consumption precisely because they were the first. Typically, it is recommended to run a dummy task before starting to measure energy consumption. It can take as long as five minutes, depending on the scenario, and can be done by executing a CPU-intensive task, such as the Fibonacci sequence.

repeat
- Despite all efforts, subsequent energy measurements will lead to different results. It is therefore common practice to repeat executions and average the results. Depending on the scenario, the number of repetitions could go up to 30. This is the typical magic number in scientific studies because it enables statistical analyses. There is no golden rule: the best method is to adjust this number along the way based on the variability of the results.

rest
- Give the device a one-minute sleep between measurements. If we repeat the same experiment 30 times with no rest in between, our CPU will probably be

warmer in the last experiment than in the first. It is important to make sure that all measurements are executed under the same conditions. Hence, it is common practice to ensure a pause/break of a few seconds or minutes between repetitions. It can be more or less dependent on your hardware or the duration of your energy test.

- In some cases, we collect energy data to compare two versions of the software when performing the same use case. If we execute version A 30 times and version B another 30 times, there is a chance that between the first and the second half of measurements, the factors affecting the energy consumption have changed. Hence, the energy consumption of the two versions will also be different because the context has changed. To mitigate this issue, it is important to mix the executions of the different versions. shuffle
- It is well established that temperature affects energy consumption. We should not let differences in temperature affect our measurements. For example, it can happen that we measure the software over different periods of the day. This means that some measurements could happen during the night, with lower temperatures, or during the day, with higher temperatures. This bias has to be avoided at all costs to avoid misleading observations. This is done by controlling room temperature and collecting temperature data alongside energy data. With temperature data, one can discard energy measurements that might be misleading. keep it cool
- Although this sounds obvious, it tends to be challenging. If we manually run the software and test its features, we are introducing a new variable that may affect energy consumption. One cannot be sure that the interactions are exactly the same. automate executions

All these practices help improve the reliability of energy data collected. However, for practical reasons, they need to be taken with a grain of salt. There might be scenarios where overlooking some of these practices will not lead to significantly different results. As always, perfect is the enemy of good. Measuring something, even with flaws, is better than not measuring anything at all.

11.4 Best practices for green software development

Above, we discussed the difficulties inherent in measuring the energy consumption of a software application. It requires a great deal of effort and time to assess whether a given use case is energy efficient. In practice, developers may not be able to afford to follow such a meticulous process to collect energy. Hence, other alternatives ought to be considered.

This is where best practices or guidelines can play a major role. In other words, you do not need to measure every single line of code to have a certain degree of confidence that the code is energy efficient. As long as it follows energy efficiency

guidelines, energy measurements can be reserved for cases where it is deemed necessary.

Guidelines can provide advice at several levels of the software development process: at the code level – with code patterns, testing; at the server level – with regard to the deployment strategy, servers used, hardware; at the project management level – with practices ensuring skill diversity, knowledge transfer, refactoring iterations; and so on. In this section, we delve into existing design patterns for energy efficiency, we look at how the choice of programming language can affect the energy efficiency of a software system, and we showcase how the energy consumption of software can be addressed in the cloud.

11.4.1 *Green software design patterns*

It is not always necessary to measure the energy consumption of an application to understand that it will probably have issues in this area. If software developers are aware of design patterns for energy efficiency, they can already anticipate these scenarios. The good thing about energy issues is that they tend to be recurrent and will probably apply to different applications.

energy patterns

Design patterns for energy efficiency are typically known as energy patterns. They provide developers with advice on the best way to design their software code so that it runs with the minimum energy consumption. They are typically defined by (at least) three components: *context*, explaining where the pattern can be used; *solution*, explaining what can be done to improve energy efficiency in that context; and *examples*, with a few instances of software code where the pattern was applied. This proved to be useful when designing mobile applications (Cruz and Abreu, 2019).

Despite energy patterns being a useful resource for developers that want to build energy efficient tools, they can also be a wonderful tool for knowledge transfer. Imagine, for example, that in a software organisation with several teams, there is only one developer that is an expert in green software. This developer cannot be involved in too many projects at the same time, so most projects will not have anyone with expertise to improve the energy efficiency of their code. However, the developer is keeping a log tracking the main energy efficiency improvements that they have been making in their apps. At the end of the week, the developer documents these improvements and adds them to a catalogue of energy patterns that is accessible by other developers from other teams. These developers will then be able to learn from the expertise of the green software developer. Assuming that there is enough freedom for developers to explore this catalogue and apply some of the patterns in their own code, eventually the software organisation will have more developers with knowledge on energy-efficient coding practices at a very reasonable cost. In other words, education is an important aspect of green software and there are many easy ways to facilitate this.

11.4.2 *Green programming languages*

Green software is all about choices and trade-offs. When choosing the programming language for their next project, software developers have to consider a number of factors: experience of the team, code readability, the language community and ecosystem, existing libraries, and so on. To add to that, a group of researchers have recently studied the energy consumption of different programming languages (Pereira, 2021). As it turns out, the choice of language can have a massive impact on the energy consumption of the software.

The group of researchers took the existing benchmark for programming languages, called 'The Computer Language Benchmarks Game', and collected energy consumption metrics. The benchmark is used by experts from all over the world to rank programming languages according to their time and memory efficiency. The same tasks are solved across different programming languages and then results are compared.

When analysing the energy consumption data, the results were astonishing: popular programming languages such as Python were observed to be 76 times less energy efficient than top performing languages such as C. This is particularly interesting as Python is becoming more and more popular given its ease of use and available libraries for AI-based software development.

It is important to note that, in the case of using Python for AI development, most of it relies on third-party libraries that are often developed in C. This means that although we are coding in Python, our final software is probably using other programming languages under the hood and the overall energy efficiency is much better than one would expect. Nevertheless, it is important to be aware of these details – energy efficiency is not a given and our choices can make a big difference.

11.4.3 *Carbon-aware data centres*

Data centres continuously run a large number of complex everyday software tasks – for example, the so-called 'cloud' is nothing more than several large-scale software systems running in data centres. Naturally, it cannot be done without a massive ecological footprint from powering and cooling down all the servers running in the data centre. To reduce carbon emissions, many data centres are opting for renewable energy. However, renewable energy is not always available. Depending on demand and the availability of clean energy in the grid at a particular time, data centres may have to resort to less clean power sources. This is more severe during peak hours, when there is high demand and servers have to run at full capacity, requiring more energy to power the data centre.

Some of the tasks running during peak hours are urgent and need to be immediately executed. For example, when a bank system is processing a payment from a grocery store. Any small delay leads to poor customer experience. However, there are other tasks that do not really need to be executed at a precise second or minute.

For example, many AI training tasks take several hours to execute. In those cases, it might be okay to wait a couple hours before performing the task.

Software tasks that have flexibility in their execution time offer an excellent opportunity to reduce carbon emissions. For example, if the carbon intensity from the power grid is high – i.e. there is not enough clean energy in the grid – we can simply wait until there is enough clean energy before we execute that task.

This simple idea makes it easier to make sure a data centre runs mostly on clean energy. It has been tested in a few contexts. However, it still poses a few challenges: estimating the carbon intensity of the grid for the next few hours, predicting how complex a given task is, predicting how much time it takes to finish, and defining how much time it can be delayed. Making incorrect estimates of these numbers can lead to suboptimal results. For example, if we predict that the sun will be bright later in the day – meaning that solar panels will produce clean energy – and then it turns out to be cloudy, delaying tasks could be emitting more carbon than we anticipated. Nevertheless, preliminary results show great potential in this strategy.

With simple ideas like this, the carbon footprint of software systems running in data centres can be massively improved. However, the carbon footprint of software is always being challenged with new cutting-edge technologies that always require more energy and resources. One example is the new hype for AI technologies, which require a massive amount of data and computational resources.

11.5 Green AI

The advent of artificial intelligence (AI) is making the new generation of software intelligent. Software systems feature components that perform tasks that until now have only been accomplished by humans. It takes software use cases to another level, e.g. by enabling them to generate text, images, and recognise objects, faces, etc. It can even perform certain tasks better than humans – for example, AI models are widely used to detect fraudulent online transactions.

machine learning

Machine learning (ML), the most popular form of AI, consists of feeding large amounts of data into a computational pipeline that will process it and extract patterns in order to generate a model – a task known as training. This model can then be used to take inputs from users and return meaningful answers – a task typically known as inference. Of course, these models are mostly useful when integrated with software. For example, the model itself does not provide a graphical user interface that allows users to interact with it. So, in general, AI models are being used without users even being aware of it. Behind every bank transaction, social media post, etc. there are multiple models running in the background.

Despite the numerous benefits of adding AI models to software systems, there is an inherent cost that needs to be addressed: energy consumption. These models require a great deal of energy to be developed and executed. For example, the

popular large language model ChatGPT consumed 550 tonnes of CO_2-eq to be trained (Patterson *et al.*, 2021). This is equivalent to driving 1000 cars for 1000 kms. To make matters worse, this is only related to a single training of the model – the cost of experimenting and tuning the training strategy took several iterations before converging to the final one. If we factor in the cost of running the model – i.e. interacting with the chat – this number will probably be astronomical. According to the latest numbers, the cost of running ChatGPT for a year is equivalent to 25 times the cost of training it (Chien *et al.*, 2023). When we think that AI is being considered a competitive technology for every business, it is not too big a leap to imagine these numbers quickly escalating.

This means that modern AI, despite being a powerful technology to combat climate change, has a carbon footprint of its own that cannot be ignored. Hence, when assessing the quality of a model, the answer can no longer be exclusively about its ability to come up with correct answers. Other metrics need to be looked at. AI is therefore becoming an important part of building sustainable software systems. It calls for a redesign of the processes we have for developing and monitoring AI systems, to ensure not only green software, but also green AI.

11.5.1 *A quality perspective on green AI*

Many have analysed the differences between rule-based software systems and (self-) learning software systems (Heck *et al.*, 2021).

With respect to the energy consumption of AI-enabled systems, four differences stand out:

1. AI-enabled systems consist of code, data and models (see Figure 11.4). This means we need to incorporate performance efficiency of data and models too.
2. Data is considered the new gold due to the hype surrounding data analytics in general and AI in particular. This encourages organisations to store huge amounts of data, with a considerable environmental cost, even if they are not sure they will use this data in the future.
3. AI-enabled systems are very computer-intensive because of the complicated mathematical calculations that are behind the machine-learning algorithms and the large datasets that they need to be applied to.
4. There is a focus on optimal models (see Figure 11.4), where stronger results are 'bought' by massively increasing computational power. Schwartz *et al.* (2020) call this 'Red AI'.

So green AI is essentially green software on steroids. That is why it emerged as a separate research field and is being addressed in many industry standards and guidelines, including the upcoming EU AI Act. For example, the EU guidelines for trustworthy AI (EU, 2019) list seven requirements, including 'Societal and environmental well-being'. According to this requirement 'AI systems promise to help tackle some of the most pressing societal concerns, yet it must be ensured that this occurs in the most environmentally friendly way possible. The system's

trustworthy AI

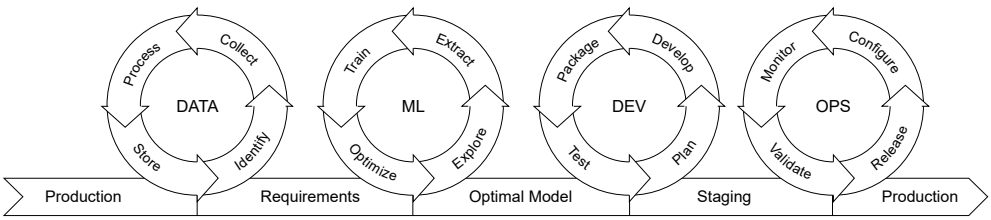

FIGURE 11.4 Development process for AI-enabled systems, including a data and model (ML) loop

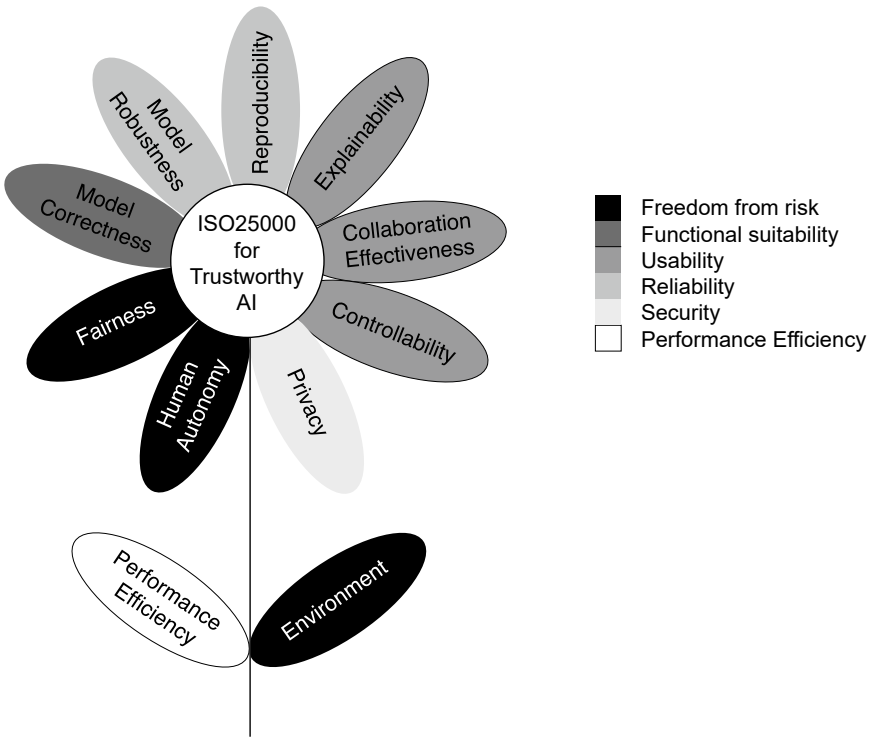

FIGURE 11.5 Green AI at the root of Trustworthy AI

development, deployment and use process, as well as its entire supply chain, should be assessed in this regard, e.g. via a critical examination of the resource usage and energy consumption during training, opting for less harmful choices. Measures securing the environmental friendliness of AI systems' entire supply chain should be encouraged.'

Heck (2022) analysed the quality characteristics of AI systems and concluded that nine properties need to be added to ISO 25000: the leaves of the flower in Figure 11.5. However, from the description in the EU guidelines we can see that in fact green AI quality characteristics are the same as green software quality characteristics:

performance efficiency and environmental risk mitigation. According to those same guidelines, environmental well-being is a necessary condition for trustworthy AI. This vision is depicted in Figure 11.5, highlighting the importance of green AI in the development of future AI-enabled software systems.

As described in Figure 11.4, the AI development process covers a wide range of things that need to be addressed before converging to a fully functional AI system. Such a complex pipeline implies that to make green AI systems we need to address energy efficiency at all frontiers.

11.5.2 *Best practices for green AI*

Green AI as a field is still in its infancy but there are already some interesting approaches that deserve a mention. One is data simplification. Not all the data we collect brings something worthwhile to be learned. The way we have been dealing with issues in data quality is by increasing the size of our data. This way, any issues in the data are attenuated by other data points. However, if we increase the size of the data, we also increase the computation required to train the model. As shown by Yarally *et al.* (2021), simple techniques for feature selection or data size reduction can have a massive impact on energy consumption.

data simplification

Other ingenious techniques rely on simplifying the model. Popular techniques of model simplification are quantization, pruning, and distillation (Van Steenweghen, 2023). Quantization consists of changing the data types used at model training – e.g. by reducing the number of bits used. Pruning consists of removing parts of a neural network that are not relevant for the performance of the model. Distillation consists of learning a new smaller model – also known as the student model – based on the knowledge of the original large model – also known as the teacher model. The downside is that some of the model simplification strategies can make model training more energy intensive, as they require extra computation for training. The benefits are mostly achieved in terms of the energy consumption of using the model. Another downside is the fact that applying some of these techniques is often not straightforward and requires domain expertise. These techniques come in different flavours and require tuning before they can produce improvements in energy efficiency. A good example of this is the project LLaMA.cpp: a port of Meta's large language model LLaMA that aims at running in low-powered devices such as a smartphone but running several model simplification techniques.

simplifying the model

Another interesting angle is the pareto between the training pipeline and the training hardware. When they want to train their models, AI practitioners often opt for the best server available. This is a natural choice, because you want to use the best state-of-the-art tools available. However, as usual, we need the best tool for the right job. When we select the most powerful hardware, this does not necessarily give us better results. Research has shown that, depending on the model, a hardware setup with lower specs can lead to better energy efficiency without hindering accuracy metrics (del Rey *et al.*, 2023).

lower specs

Such an observation is in line with reports from Meta revealing that a vast portion of their AI systems are only using GPUs at 30% of their full capacity (Wu *et al.*, 2022). This is a major issue because it means that we need three times more GPUs to compensate for this lack of resource efficiency. Moreover, hardware has an embodied carbon footprint – i.e. the carbon emissions incurred by producing and shipping it – that is being wasted on stalled GPUs.

obsolete hardware

Another issue arising from reliance on the latest hardware is that existing hardware soon becomes obsolete (del Rey *et al.*, 2023). The latest AI libraries are quickly dropping their support for older hardware. This does not necessarily mean that the hardware is unable to perform the required computational tasks. In some cases, it simply means that the cost of providing support to old hardware has become economically unsustainable. The consequences of this problem are twofold: 1) we are reducing the lifecycle of hardware without reducing its embodied carbon footprint, and 2) only practitioners that have the capacity to buy new state-of-the-art hardware are able to run the latest advancements in AI technology. This last consequence raises an issue in terms of AI democracy. We want AI technologies to be accessible to any group of tech enthusiasts and not just to a few big tech companies that have enough negotiating power and financial resources to get the hardware that runs the latest AI software.

11.6 Conclusion

This chapter lays the foundation for sustainable software engineering and all its different dimensions. The means to creating sustainable software requires a multifactorial approach that considers trade-offs between different sustainability aspects: environmental, social, individual, technical, and economic. This is easier said than done: despite being covered by industrial software quality models, designing green software systems requires a number of non-trivial techniques to test and monitor the energy efficiency of software.

In this chapter, we have seen how energy patterns and other best practices can help build energy-efficient software by design. However, the advent of AI is challenging the software industry by requiring even more computational power to run state-of-the-art intelligence. The emerging guidelines for trustworthy AI make performance efficiency and environmental risk mitigation essential requirements.

However, more work needs to be done to enable a fully green landscape in the realm of intelligent software. This is a concern that involves developers, designers, policymakers, users, and society in general. We currently rely on software for everything – as software users, we have the right to ask for transparent and trustworthy software. When we download a new app or sign up for a new service, we want to know what is behind the curtain. All stakeholders have a role to play, and little by little, step by step, we will make all software green, together.

References

Becker, C., Chitchyan, R., Duboc, L., Easterbrook, S., Penzenstadler, B., Seyff, N. and Venters, C.C., 2015. Sustainability design and software: the Karlskrona manifesto. In: 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering (Vol. 2, pp. 467–476).

Chien, A.A., Lin, L., Nguyen, H., Rao, V., Sharma, T., and Wijayawardana, R., 2023. Reducing the carbon impact of generative AI inference (today and in 2035). In: Proceedings of the 2nd workshop on sustainable computer systems (pp. 1–7).

Cruz, L. and Abreu, R., 2019. Catalog of energy patterns for mobile applications. Empirical software engineering, 24:2209–2235.

Cruz, L., 2021. Green software engineering done right: a scientific guide to set up energy efficiency experiments. http://luiscruz.github.io/2021/10/10/scientific-guide.html.

del Rey, S., Martínez-Fernández, S., Cruz, L., and Franch, X., 2023. Do DL models and training environments have an impact on energy consumption? In: 49th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), pp. 150–158.

EU, 2019. Ethics guidelines for trustworthy AI. Brussels: European Commission. https://data.europa.eu/doi/10.2759/346720.

EU, 2022. Questions and answers: EU action plan on digitalising the energy system. https://ec.europa.eu/commission/presscorner/detail/en/qanda_22_6229.

Heck, P., Schouten, G., and Cruz, L., 2021. A software engineering perspective on building production-ready machine learning systems. In: Handbook of research on applied data science and artificial intelligence in business and industry (pp. 23–54). IGI Global.

Heck, P., 2022. A quality model for trustworthy AI systems. https://fontysblogt.nl/a-quality-model-for-trustworthy-ai-systems/.

Patterson, D., Gonzalez, J., Le, Q., Liang, C., Munguia, L.M., Rothchild, D., … and Dean, J., 2021. Carbon emissions and large neural network training. arXiv preprint arXiv:2104.10350.

Pereira, R., Couto, M., Ribeiro, F., Rua, R., Cunha, J., Fernandes, J.P., and Saraiva, J., 2021. Ranking programming languages by energy efficiency. Science of computer programming, 205, 102609.

Schwartz, R., Dodge, J., Smith, N.A., and Etzioni, O., 2020. Green AI. Communications of the ACM, 63:54–63.

Van Steenweghen, A., 2023. EasyCompress: automated compression for deep learning models. Master Thesis TU Delft, the Netherlands.

Wu, C.J., Raghavendra, R., Gupta, U., Acun, B., Ardalani, N., Maeng, K., Chang, G., Aga, F., Huang, J., Bai, C. and Gschwind, M., 2022. Sustainable AI: environmental implications, challenges and opportunities. In: Proceedings of machine learning and systems, 4:795–813.

Yarally, T., Cruz, L., Feitosa, D., Sallou J., and Van Deursen, A., 2023. Uncovering energy-efficient practices in deep learning training: preliminary steps towards green AI. In: IEEE/ACM 2nd International Conference on AI Engineering–Software Engineering for AI (CAIN) (pp. 25–36).