Green software guide

Developed in partnership with the University of Leicester

Introduction

Greenhouse gas emissions from software are increasing at rapid pace, with the ICT industry set to reach a 14% share of global emissions by 2040, a sharp increase from the previous 5% in 2022. Software use is widespread and has significantly transformed almost every aspect of life and work. Software is the set of instructions, data and programs that enable technology to function. Many ICT applications require large, complex software and although software doesn't consume energy directly, it directs and influences the operation of hardware, thereby impacting hardware's energy consumption and environmental impacts. Inefficient software can have a major impact on energy consumption so consideration of sustainable software is vital to achieving more efficient hardware.

Increasing access to hardware (e.g. smartphones, tablets and laptops) provides greater access to the internet thereby reducing the digital divide and digital poverty. This increasing access enhances society's reliance and a demand for IT which is driving the creation of further applications that are larger and more complex, such as ChatGPT. This increased use of hardware is coupled with increased software use, highlighting software to be an important issue for sustainability.

Green software is considered efficient software which meets the needs of the present without negatively impacting future society. The concept of green software places paramount importance on integrating energy-efficient and waste-reduction patterns, due diligence practices and methodologies across the entire software delivery cycle. Green software offers substantial potential to curtail carbon emissions and alleviate the financial burdens tied to outdated, ineffective legacy technologies and workflows.

Green software purchasing

Sustainable procurement involves procuring goods and services to fulfil objectives whilst reducing negative environmental and social impacts. Sustainable purchasing of software includes development, deployment, and management. When procuring software, it is important to consider the following:

- Determine whether your supplier has green credentials in software and where they are along the maturity curve.
- Determine whether your supplier has sustainability standards and policies for software design and development, software energy efficiency, software carbon intensity and accessibility features.

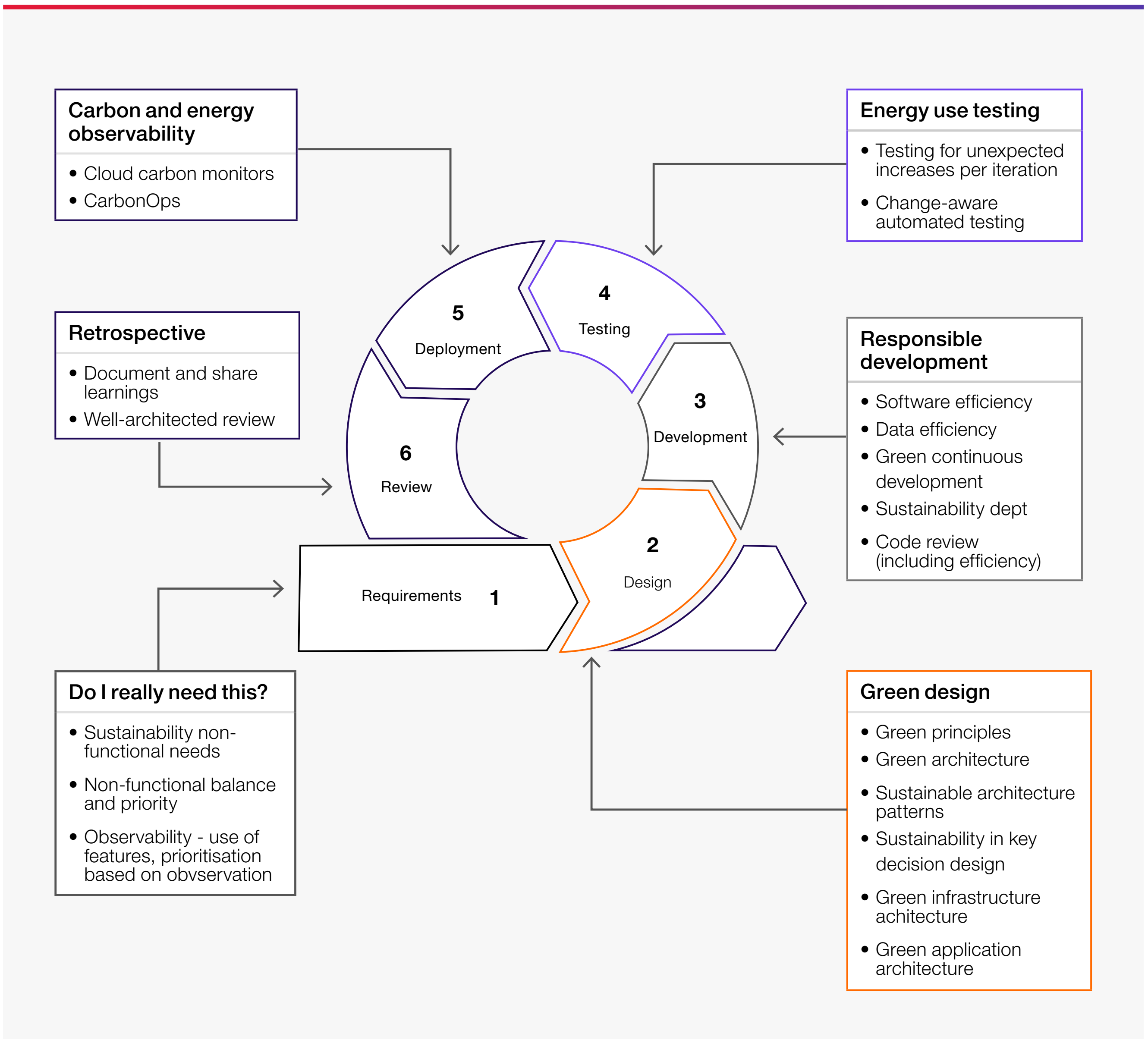
Software design and development

It's important to ensure that software is designed to optimise energy consumption. The impact of software design on overall energy consumption is significant, as evidenced by an increasing number of real-world examples.

Harvard Case Study

The design and development of software can be energy intensive. For example, researchers at Harvard trained an AI model to classify flowers using a small, publicly available dataset. The AI model achieved an accuracy of 96.17 percent in classifying different species with only 964 joules of energy. But to achieve higher accuracy the system consumed significantly higher energy—to gain a 1.74 percent increase in accuracy. The energy consumption increased significantly to 2,815 joules and a further increase in accuracy demanded increased energy consumption.

Development of green software spans the entire lifecycle of software development, operation and disposal (reuse). Green software design and development requires a mindset shift that integrates sustainability by recognising its environmental impacts across its lifecycle. Considering where to include green software in your delivery lifecycle can be challenging. We've developed the infographic below as a basic guide to green software in the delivery life cycle.


Carbon and energy observability
• Cloud carbon monitors
• CarbonOps
Retrospective
• Document and share learnings
• Well-architected review
Do I really need this?
• Sustainability non-functional needs
• Non-functional balance and priority
• Observability - use of features, prioritisation based on obvservation
Energy use testing
• Testing for unexpected increases per iteration
• Change-aware automated testing
Responsible development
• Software efficiency
• Data efficiency
• Green continuous development
• Sustainability dept
• Code review (including efficiency)
Green design
• Green principles
• Green architecture
• Sustainable architecture patterns
• Sustainability in key decision design
• Green infrastructure achitecture
• Green application architecture
1 Requirements
2 Design
3 Development
4 Testing
5 Deployment
6 Review

Architecture

Architecture pattern refers to the structure and organisation of the software systems. Selecting the appropriate architectural pattern and deployment model for software has a significant impact on the sustainability of software in terms of structure and how resources are used. Architecture deployment involves installing architectural patterns into a physical environment. The impact of changes to architectural patterns can include:

- Resource efficiency
- Scalability
- Performance optimisation
- Modularity and maintenance
- Longevity and lifecycle management
- Energy efficiency
- Carbon intensity
- Cloud and virtualisation
- Data management decommissioning
- Legacy transformation

Deployment models such as architecture impact the sustainability of software by influencing resource use, scalability, energy consumption, and efficiency. Identifying and recognising one deployment model as the most sustainable is problematic; the sustainability of a deployment model will vary depending on the specific needs and requirements of the software, the energy consumptions of the hardware options and the lifecycle of the software.

The environmental impacts of deployment models vary. On-premises deployment, for example, requires the purchasing and maintenance of physical hardware which increases demand for raw materials and has a negative environmental impact. Alternatively, cloud deployment measures are typically viewed as more sustainable due to scalability, however, can be energy intensive, increasing costs and environmental impacts. It is important to make decisions relating to sustainability in the context of each software and external influencing factors.

Features such as the maintainability of software are a key factor in sustainable software development and deployment. Software which is easy to maintain has better energy performance, removing the need to re-engineer the whole software. Re-engineering software requires significant resources, so maintainable deployment positively impacts resource efficiency relating to energy consumption, time, and user experience. Software architecture can also enable a more effective user feedback system, improving the end user's experience and software development. For example, bugs can be easily fixed, and new features seamlessly added.

Microservices and cloud-native architecture are two architectural patterns leading to improved sustainability of software. Microservices can decompose software into small independent services. Cloud-native architecture can react to events, rather than polling for data. Reusable application programming interface (API) and administrator limits can help achieve the goal of more sustainable software. Reusable APIs can help to reduce and regulate code that needs to be written to achieve a desired functionality. This restriction can save energy. Limits can also prevent unnecessary processes or services from being run which waste energy.

Top Tips

- Use tools and resources which support developers to measure energy. CGI's DataTwin360 determines real-time energy measurements from Cloud systems within their applications to provide a view of efficiency and emissions.
- Monitor real-time power consumption during development through techniques such as dynamic code analysis. This will identify SQL query injection, long input strings, negative and large positive numbers and unexpected input data to determine gaps.
- Consider data usage during the design process by managing the lifecycle of stored data e.g., compressing and aggregating data, minimising data exchange and adopting efficient cache policy.
- Remove and refactor unused features to improve energy efficiency, and make software easier to maintain.
- Limit the computational accuracy of the application to the desired level, considering what is appropriate for operational requirements.
- Remove loops which do not achieve the intended purpose.

Software optimisation

As a result of the industry's limitless software capabilities, many of our codes today are inefficient, oversized and bloated with unused features.

Software code optimisation is a key step in achieving energy efficiency, high-performance and sustainable goals. Software code written by the application developers goes through the process of compiling. A compiler converts the code of high-level programming into a computer-readable low-level programming language, and the process allocates memory which uses run-time resource-efficient coding can help to reduce the run-time cycles by optimising code usage. The environmental impact of running inefficient code, although not always obvious, amplifies the carbon footprint of the company developing inefficient code.

Benefits of code optimisation

Resource Management – Optimised code makes effective use of the resources it has been presented with. Code requires memory to store and run. An industry standard sustainable practice of green coding is "reusability". A written code is only sustainable if it can be reused in as many scenarios as possible.

Energy consumption – Optimised code consumes less energy. The code that has been deployed in a cloud infrastructure consumes a set amount of energy it needs to run, and if the deployed code has energy efficient algorithms (such as minimal loops, appropriate data structures, energy-efficient libraries), the code consumes less energy.

Scalability – Optimised code is easily scalable. During software development, as it goes from the conceptual stage to the deployment stage, there must be instances where the code needs to accommodate new operations. With new operations, the code utilises more resources, hence more energy is consumed. An optimised code which uses modular techniques or compartmentalisation can accommodate scalability issues.

During this phase sustainable code is crucial. Developing software code that uses less energy can lead to significant emissions reductions, mainly when deployed at scale.

Five principles of code optimisation

- Write code that is more efficient
- Consume electricity with low-carbon intensity
- Reduce the amount of data transfers between the networks and servers
- Maximise energy efficiency of hardware
- Measurement and optimisation

Software management

Green software usage, management, and support are crucial aspects of maintaining a healthy and productive software ecosystem. The development and operation of software involves several supporting systems, which have an impact on energy consumption. Supporting systems include:

- **Continuous delivery pipelines**
- **Automated testing**
- **Idle compute**
- **Network transfer**
- **Storage**
- **Monitoring**
- **Logging**
- **Backups**
- **Failover/redundant resources**

Each of these systems may significantly contribute to overall sustainability depending on the type, data policy, and design of your software—for example, long-term data retention on large datasets, network transfer for streaming and energy consumption during idle compute.

Sustainable software management:

Dependency management: Regularly update dependencies to ensure the software stays secure and compatible with the latest technologies. Automated dependency management tools can help streamline this process.

Continuous integration/continuous deployment (CI/CD): Implement CI/CD pipelines to automate software build, testing, and deployment processes. This reduces manual effort, improves software quality, and accelerates release cycles.

Monitoring and performance optimisation: Monitor software performance and resource utilisation to identify bottlenecks and areas for optimisation. Proactively address issues to ensure efficient operation and user satisfaction.

Security: Implement robust security practices to protect against vulnerabilities and cyber threats. This includes regular security audits, patch management, and adherence to security best practices.

Scalability: Design software with scalability in mind to accommodate growth in user base and data volume. Architectural decisions such as microservices, containerisation, and cloud-native technologies can facilitate scalability.

Sustainable software support:

Community engagement: Foster a vibrant user community around the software through outreach, collaboration, and user feedback mechanisms. Engaged communities can provide valuable support, contributions, and advocacy.

Issue tracking and resolution: Maintain an issue tracking system (e.g., GitHub issues) to manage bug reports, feature requests, and other user feedback. Prioritise issues based on severity and impact and provide timely resolutions.

User feedback and iterative development: Solicit and incorporate user feedback into the software development process. Regularly release updates and new features based on user needs, priorities and energy consumption.

Long-term maintenance: Plan for long-term maintenance and support of the software, including versioning, backward-compatibility, and end-of-life policies. Consider establishing a maintenance team that is trained in carbon accounting or engaging with third-party support providers if needed.

Knowledge sharing: Document lessons learned, best practices, and troubleshooting guides to facilitate knowledge sharing among support staff and the user community. This can help improve support efficiency and empower users to resolve issues independently.

By prioritising sustainability in software usage, management, and support, organisations can maximise the longevity, resilience, and value of their software assets while minimising associated risks and costs.

	Principle	Guidelines
1	**Measurement** To identify the impact of software on the environment, you must baseline the level of sustainability of your software before integrating sustainable software principles. For example, by measuring energy consumed, carbon emitted, and e-waste generated through for example incompatible software, we can identify areas that require improvement. Without measurement, identifying areas for improvement will be challenging.	Choose the relevant metrics for your software. Establish a baseline and quantify the current sustainability of your software. Follow the Plan, Do, Check, Adjust method. Identify improvement opportunities. Use the baseline to identify optimisation opportunities. Implement principles and strategies. Monitor progress - review your progress against your baseline and identify opportunities for further improvement or re-address areas that might not be performing as well as expected. Adjust your implementations through data-driven decisions. Report on progress to identify overall trends and success.

	Principle	Guidelines
2	**Efficient use of hardware** **Rationale:** Inefficient use of hardware is unsustainable as it produces a significant amount of hardware garbage, and e-waste. Efficient use of hardware is more cost effective and sustainable than purchasing new hardware.	Analyse software requirements, to identify the most sustainable options. For example, identify trade-offs between using on-premise hardware until the end of its service life in comparison to creating e-waste via opting for cloud hosting for your software. Avoid the development of software only compatible with new hardware to avoid the creation of e-waste and increasing demand on natural resources for new hardware. Ensure hardware and server capacities are maximised before purchasing new hardware. Consolidate multiple servers and storages to reduce hardware accumulation and e-waste. Utilise server virtualisation which allows multiple operating systems to run on one server.
3	**Storage optimisation** As a result of data retention policies, 95% of an organisation's data remains unused in data landfill. Significant data landfill is complex to maintain and increases economic capital required for new hardware. Additionally, the process of purchasing new and unrequired hardware is unsustainable due to the environmental impacts of hardware on natural resources.	Replication of data occurs in many systems where the same data is stored in different storages for different usages. Smarter, more sustainable software should be coded to exploit parallel, multiprocessor architectures to develop efficient use of hardware resources. Reduce unwanted redundancy of data in data lakes and data landfill to free up storage for future applications. Utilise memory leak detection tools, memory leaks require more storage space than required. Identify storage optimisation in software through code review tools. Use compression technologies to reduce the size of required data. Introduce stringent data retention policies and only archive essential data.

	Principle	Guidelines
4	**Carbon aware engineering** Building applications that are carbon aware consistently begins with software engineers engaging in carbon aware consistently and mindful programming.	Avoid feature creep – prevent the excessive use of unnecessary features within an application which extends the length of code and simultaneously increases the monetary and carbon cost of software. Be mindful of the size and quality of images, video and audio used, these can be compressed through a compression tool. Build software that requires fewer lines of code to achieve desired functionality.
5	**Energy efficient consistently** Building energy efficient software is a fundamental principle of designing and building sustainable software. Energy efficient software saves organisations money, contributes to the achievement of net zero targets and can improve user experience.	Choose hardware that is energy efficient. Optimise database queries. Avoid unnecessary iterations that increase demand on energy resources. Design efficient user interfaces. This can include being mindful of audio, video and graphic size and quality, that requires increased energy. These can be compressed through a compression tool. Lazy loading should also be utilised to avoid loading of resources until required. Implement power management features such as sleep mode to reduce energy consumption when software is idle. Implement preloading and caching, this will decrease the frequency of data being accessed from a disc.

	Principle	Guidelines
6	**Carbon efficiency** Carbon efficient software results in a reduced consumption of energy, compute power and wider materials. Carbon efficient software achieves maximum performance per unit of carbon emitted.	Algorithms must be energy efficient. Use renewable energy sources to power hardware, both on-premises and cloud. Optimise code, for example by avoiding unnecessary loops and lines of code to achieve user needs. Use sleep modes to reduce energy consumption. Take advantage of scalability if hosted on the cloud to avoid unnecessary resource use. Use containerisation technologies to isolate applications and services.
7	**User carbon awareness** Carbon aware applications are aware of the environmental footprint of the software. This awareness is necessary for both the software itself and the user to encourage efforts to reduce environmental impact. This awareness is critical for the user, as the carbon impact of software use is invisible to the user, therefore there is a disconnect between action and consequence, that if visible, would encourage behaviour change.	Demand shaping – users should strategically schedule and optimise software use to align with periods whereby the energy demand and therefore energy mix are more likely to be more renewable energy. Updates and tasks should be completed during off-peak energy times. Energy-efficient modes – users should use 'eco-modes' or energy efficient modes that run on lower functionality such as reduced brightness to minimise the energy required and carbon emitted from powering the software. Energy monitoring and carbon transparency – provide the user with real-time data regarding energy use and carbon emissions from software use to bridge the gap between action and consequence. Sustainability based alerts – these will make users carbon aware and encourage active energy saving.

About CGI

Insights you can act on

Founded in 1976, CGI is among the largest IT and business consulting services firms in the world.

We are insights-driven and outcomes-based to help accelerate returns on your investments. Across hundreds of locations worldwide, we provide comprehensive, scalable and sustainable IT and business consulting services that are informed globally and delivered locally.

cgi.com/uk