



Sustainability in Software Engineering Lifecycle

Yousfi Abderrahmane

Master's thesis in Industrial Engineering

Yousfi Abderrahmane,

Advisor:
Zemmouchi-Ghomari Leila

Co-advisors:
Belayadi Djahida

Academic year:
2024-2025

Abstract: As digital technologies expand, their environmental impact has become a growing concern. Sustainable Software Engineering (SSE) offers a promising pathway to address this issue by embedding sustainability principles throughout the Software Development Life Cycle (SDLC). This research explores how methods and tools from industrial engineering can be applied to software development processes to support sustainability goals. It focuses on improving resource efficiency and structuring sustainable workflows using models like GREENSOFT. The study aims to equip software practitioners with practical, scalable strategies drawn from industrial engineering to guide responsible and efficient development. By adapting these methods, software and industrial engineers can work together to reduce CO2 emissions, improve energy efficiency, and integrate environmental concerns without compromising software functionality. This approach provides actionable strategies tailored to industrial engineering contexts. It contributes to a broader understanding of how sustainability can be systematically embedded into software design and engineering processes to meet the dual goals of technological advancement and ecological responsibility.

Key-Words: sustainability, software, lifecycle, CO2 emission, sustainable software engineering, SDLC, LSD, LCA

Contents

1	Introduction	2
2	CO2 Emissions and IT	2
3	Sustainable Software engineering	3
4	Engineering Methods For a Sustainable SDLC	6
5	Discussion	9
6	Conclusions	10

1. Introduction

The software industry plays an increasingly central role in shaping modern society. Software systems are deeply embedded in everyday life, from powering essential infrastructure to enabling digital transformation across industries. However, as software grows in scale and complexity, so does its environmental footprint. Rising energy consumption, hardware demands, and digital waste have become critical challenges within the broader field of information and communication technologies (ICT) [1, 2]. These concerns have sparked growing interest in Sustainable Software Engineering (SSE), a field focused on minimizing the environmental, social, and economic impacts of software across its entire lifecycle [3].

While sustainability in software development is gaining attention, most existing studies focus either on high-level strategies or on isolated technical optimizations. What remains missing is a structured and actionable approach that bridges strategic direction and practical implementation, while enabling continuous assessment of sustainability goals [4, 5]. In this context, integrating industrial engineering methods into the software development process presents a promising yet under-explored opportunity. These tools, commonly used in manufacturing and process optimization, offer practical techniques for embedding sustainability into software workflows but have rarely been applied systematically in the context of the SDLC.

This paper aims to address that gap by proposing a cross-disciplinary approach that combines industrial engineering tools with software development practices. It introduces a set of methods, such as Lean Software Development, Value Stream Mapping, and Life Cycle Assessment, alongside measurable sustainability metrics, including energy consumption, CO2 emissions and resource efficiency. Together, these elements provide a framework for guiding and evaluating sustainable software development in a way that is both practical and aligned with real-world needs.

The research begins by assessing the environmental impact of IT systems, particularly their role in CO2 emissions, while also recognizing the potential of digital technologies to reduce emissions in other sectors. Then, it introduces the foundational concepts of SSE, including key definitions and the SDLC as a framework for integrating sustainability. Building on this foundation, the study examines a set of industrial engineering tools and analyzes their contribution to sustainability within each phase of the software lifecycle. In parallel, it proposes a set of sustainability metrics that can be used to assess and improve the environmental performance of software systems. By integrating tools, methods, and indicators, this research aims to support the development of more responsible and resource-efficient software systems.

2. CO2 Emissions and IT

As digital technology becomes deeply integrated into everyday life, the IT sector's environmental impact is growing rapidly. While IT enables efficiency and productivity, it also requires significant energy and resources, which results in CO2 emissions. In this section, we'll examine the CO2 emissions associated with IT and the potential CO2 savings that can be achieved through IT.

2.1. CO2 Consumption Caused by IT

Every time we use digital tools, whether browsing the internet, streaming videos, or storing data in the cloud, there is an environmental impact. The IT sector, which includes data centres, networks, and devices, requires energy to operate. If it is not from renewable sources, this energy leads to the production of CO2. The growth in data centres, which power the internet and store massive amounts of data, is one of the most significant contributors to IT-related carbon dioxide emissions. Some examples of this impact include :

- Data centres, the backbone of digital infrastructure, are a significant source of CO2 emissions. The energy used by data centres globally accounts for approximately 1% of total electricity consumption, with predictions that this could rise to 8% by 2030 if no changes are implemented [1]. A large data centre can consume the same amount of power as a small city, and cooling systems for these facilities require significant energy to maintain the ideal server temperature.
- The Internet and telecommunications networks are central to our digital lives and are part of the ICT (Information and Communication Technology) sector. These networks enable data-intensive activities, such as cloud services, and are highly energy-demanding. Every gigabyte of data transferred consumes approximately 5 kWh of electricity, producing around 1.5 kg of CO2 emissions if the energy source is coal-based [2]. Mobile networks, fibre optics, and satellite communications also require extensive energy resources to support reliable, high-speed data transfer and connectivity. As global demand for online services grows, energy consumption in ICT increases as well. Studies estimate that if growth continues unchecked, ICT could contribute to over 14% of global CO2 emissions by 2040 [6].
- Devices such as computers, smartphones, and tablets contribute to CO2 emissions, mainly when not manufactured or disposed of sustainably. Studies show that producing a desktop computer emits between 120 and 250 kg of CO2, while using it adds another 50 to 175 kg of CO2 per year. Most desktops have a lifespan of 3 to 6 years, making their total impact significant [6].

2.2. CO2 Savings from IT

While IT contributes to CO2 emissions, it also has significant potential to reduce emissions across other sectors. IT can enhance industrial operations, transportation, and energy management efficiency, promising a more sustainable future [7]. Some studies include:

- IT has the power to optimise power grids [8], dynamically matching supply and demand to reduce energy waste. It has been demonstrated to reduce energy consumption by up to 10% in certain regions [9], highlighting the transformative potential of IT in combating climate change.
- Digital Transformation in Transportation: IT solutions in transportation, such as intelligent traffic systems and fleet optimisation, can reduce fuel consumption. These systems have the potential to reduce CO2 emissions in urban areas in the future. Ride-sharing applications, autonomous vehicles, and telecommuting have also reduced emissions [7].
- Smart Manufacturing: Iot sensors and predictive maintenance allow factories to operate more efficiently. Manufacturers can reduce energy use and prevent costly equipment breakdowns by monitoring machinery and optimising processes. This approach, often referred to as lean manufacturing, has the potential to reduce operational CO2 emissions [7, 8].

As Figure 1 illustrates, IT contributes significantly to CO2 emissions through data centres, networks, and device manufacturing. However, IT also offers solutions for CO2 savings, particularly in the power management, transportation, and industrial sectors. By implementing these strategies, IT can help reduce its environmental impact and play a crucial role in achieving global sustainability goals.

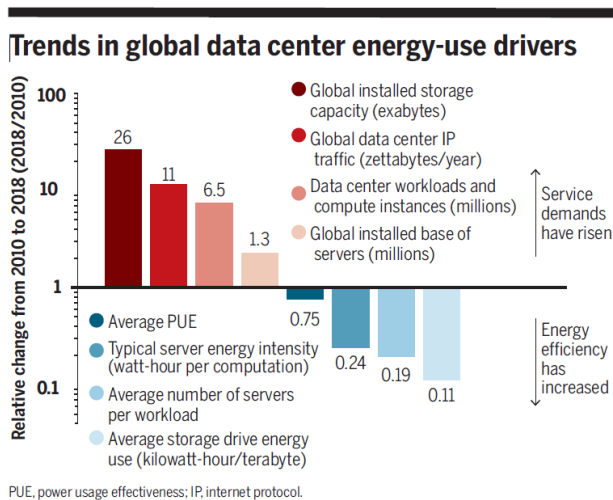


Figure 1: Data centers' energy-use [1]

3. Sustainable Software engineering

As software reliance continues to grow, so does its impact on the environment, economy, and society. While IT is crucial in enabling sustainability, software must be developed with sustainability in mind. Sustainable Software Engineering (SSE) designs and manages software systems to align with sustainability goals [10]. This section introduces the foundations of SSE by first defining key terms such as Green and Sustainable Software and Green and Sustainable Software Engineering. It then outlines the structure and phases of the Software Development Lifecycle (SDLC), which provides the context for applying sustainability principles. Understanding the SDLC is essential, as it is the operational framework through which sustainability can be integrated. We also explore how sustainability is addressed across software development's environmental, economic, social, and technical dimensions. It concludes with a presentation of the GREENSOFT Model, a structured framework that supports the integration of sustainability throughout the software lifecycle. Together, these elements offer a clear path for embedding sustainability in software engineering to maintain both performance and long-term viability.

3.1. Definitions

Definition 1 : 'Green and Sustainable Software' [3, 4] Naumann et al. have defined green and sustainable software as software designed to minimise both direct and indirect negative impacts on the economy, society, individuals, and the environment throughout its development, deployment, and use. It also aims to contribute positively to sustainable development. Achieving this requires developers to know the software's potential effects and integrate sustainability considerations into the development process. By doing so, sustainability becomes a manageable part of the workflow, enabling architects and developers to make informed improvements to the software.

Definition 2 : 'Green and Sustainable Software Engineering' [4, 11] According to Komeil et al. Green and Sustainable Software Engineering is developing software through a process that actively considers environmental sustainability. It involves continuously assessing and documenting the software's positive and negative impacts on sustainable development across its entire lifecycle, using these insights to guide ongoing optimisation.

These definitions highlight the distinction between sustainable software as a product and sustainable software engineering as a process. The first definition focuses on the software, emphasising its direct and indirect impacts on various sustainability dimensions, including economic, social, and environmental factors. It stresses the importance of minimising harm while potentially

contributing to sustainable development. The second definition expands on this idea by framing sustainable software engineering as a continuous and systematic approach. It underscores the need to assess, document, and optimise software throughout its entire lifecycle, ensuring that sustainability is a constant consideration rather than a one-time goal.

3.2. The Software Development Life Cycle

Knowing the basics of the Software Development Lifecycle (SDLC) is essential to understanding sustainable practices in software development. The SDLC is a structured process that guides software development from the initial idea to its final delivery [12]. It provides a clear framework for planning, creating, testing, and deploying software solutions, ensuring that each stage is well-organised and efficient. The success of a software project largely depends on the chosen SDLC model. This process enables developers to track the current phase of a project, allocate necessary resources, and prepare for upcoming stages [12, 13]. The SDLC is typically divided into six key phases to manage software projects systematically and effectively. These phases are outlined below and illustrated in Figure 2 :

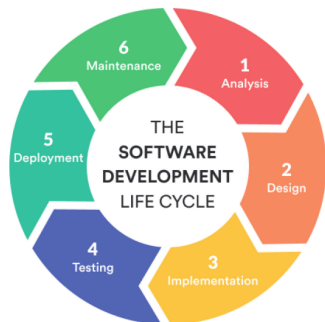


Figure 2: A Typical SDLC [14]

The Software Development Lifecycle (SDLC) comprises several essential phases that guide the development process from inception to completion. Each phase has a specific purpose and plays a vital role in ensuring the success of a software project [12, 14, 15] :

Requirement Gathering and Analysis: This is the first phase, where customers and stakeholders collaborate to understand the project’s goals and collect detailed requirements. Both functional (what the software should do) and non-functional (performance, security) requirements are identified and analysed to provide a clear foundation for the project.

Design: In this phase, the team creates the system’s architecture based on the collected requirements. This includes designing database schemas, determining hardware and system needs, analysing features, and crafting the user interface. A prototype is often developed during this stage to validate the design with stakeholders and ensure it meets the identified needs.

Implementation or Coding: This is where the actual development begins. The team writes the software code, ensuring it aligns with the system design, and conducts unit tests to verify the functionality of individual components. Once the coding is completed, the software moves to the testing phase for further evaluation.

Testing: involves evaluating the software to identify and fix bugs or issues. Various tests, such as system, integration, and user acceptance testing, are conducted to ensure the software functions as intended. Once all issues are resolved, the product is ready for deployment.

Deployment: In this phase, the software undergoes a final review, and any necessary modifications are made. Afterwards, it is deployed to the production environment, making it accessible to end users.

Maintenance: The final phase focuses on maintaining the software by addressing user-reported issues and implementing necessary updates. Fixes can be prioritised based on urgency and importance to ensure the software remains functional and up-to-date.

The SDLC provides a structured approach for managing software projects from initial concept to deployment and ongoing maintenance. Each phase is critical in ensuring software is developed efficiently and meets user expectations while remaining maintainable and functional over time. Understanding this lifecycle is essential for identifying where and how sustainability considerations can be integrated into software processes. Building on this structure, the following subsection shifts focus to SSE, exploring how sustainability can be applied not as a separate concern but as an integral part of the software life cycle.

3.3. Sustainability in Software Engineering

Sustainability in software engineering refers to designing, building, and maintaining software systems to balance current needs with long-term viability, ensuring minimal environmental, economic, and social impact [3, 10, 11]. A synthesising approach, informed by key perspectives, leads to these comprehensive aspects:

Environmental Impact : Sustainable software development minimises CO2 emissions by reducing energy consumption and avoiding excessive hardware demands. This helps conserve the environment while ensuring efficient software performance [1].

Economic Efficiency : Sustainable software emphasises affordability through maintainable and scalable systems. Preventing costly redesigns and overhauls ensures long-term financial benefits for organisations [5].

Social Responsibility : Sustainability in software prioritises inclusivity and accessibility. It aims to create user-friendly systems for all communities, promoting equity and ethical standards in digital tools. [5].

Technical Longevity : Sustainable software ensures its systems are maintainable, modular, and scalable.

Adapting to technological changes and being well-documented prevents obsolescence and supports long-term usability [5, 16].

Sustainable software development is the practice of creating systems that are environmentally conscious, economically viable, socially equitable, and technically enduring. By integrating these principles, developers can create software that meets immediate requirements and supports long-term growth, adaptability, and a minimal ecological footprint, providing reassurance for the future.

SSE's key aspects include designing efficient and scalable software to handle growing demands and making it modular for easy updates and integration. These practices help extend the software lifecycle, minimising electronic waste and supporting sustainable development. Sustainability metrics are crucial in assessing whether the software development lifecycle (SDLC) aligns with environmental, economic, and social sustainability goals. These metrics help quantify software's impact at different stages, allowing stakeholders to optimise development practices and reduce unnecessary resource consumption.

3.4. The GREENSOFT model

The GREENSOFT Model [3] is a framework designed to guide stakeholders (see Figure 3), such as software developers, administrators, and users, in making software more sustainable throughout its lifecycle. At its core, the model aims to minimise the first-order effects, including energy consumption, hardware demands, network bandwidth usage, and physical packaging, which directly impact resource consumption. It also considers second-order effects where the software's use can optimise processes in other industries, such as innovative heating or logistics, indirectly reducing environmental and resource footprints. Finally, third-order effects, such as rebound effects, are known to be more challenging to predict. These effects could result from efficiencies that accidentally increase consumption elsewhere.

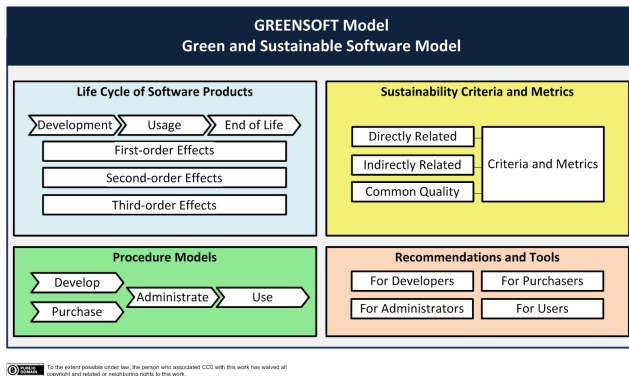


Figure 3: the GREENSOFT model [3]

The GREENSOFT model is structured around four key components:

Life Cycle of Software Products: This section adopts a holistic "cradle-to-grave" approach, assessing the environmental, social, human, and economic impacts of software throughout its entire life cycle, from development to disposal. The results of this process can be used to optimise software sustainability or compare products.

Sustainability Criteria and Metrics: This component offers tools for evaluating software sustainability, incorporating direct (e.g., energy usage) and indirect (e.g., broader societal impacts) criteria. It uses standard software quality metrics and concepts borrowed from life cycle assessments.

Procedure Models: These models integrate sustainability considerations into various stages of software management, including development, acquisition, maintenance, and user support.

Recommendations and Tools: This section offers practical guidance and tools for stakeholders, enabling them to effectively incorporate sustainable practices into the development, purchasing, and use of software. The GREENSOFT Model classifies sustainability metrics into three main categories (see Figure 4): standard software quality metrics, directly related sustainability metrics, and indirectly related sustainability metrics.

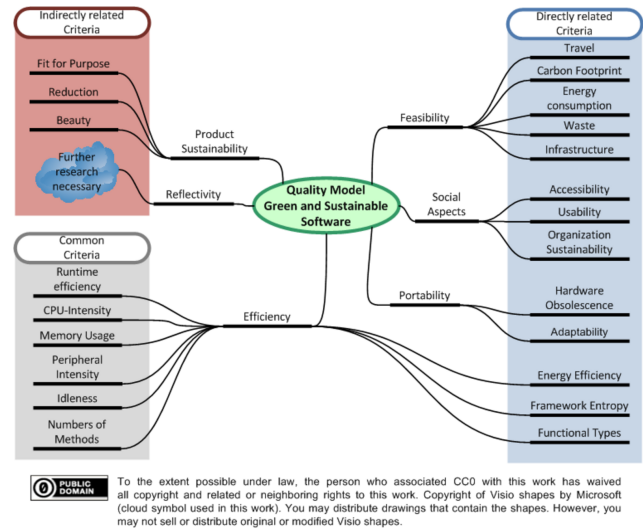


Figure 4: The GREENSOFT model sustainability metrics [3]

Standard software quality metrics include traditional performance indicators such as maintainability, portability, and reliability. These metrics indirectly contribute to sustainability by improving efficiency and extending the software lifecycle.

Directly related sustainability metrics measure energy consumption, CPU and memory usage, and storage efficiency. Software's high energy consumption can lead to excessive power usage in data centres, cloud environments, and user devices, significantly increasing carbon emissions. Monitoring these metrics enables organisations to identify inefficiencies and optimise energy usage through improved coding practices, resource-efficient algorithms, and cloud-based solutions that leverage renewable energy.

Indirectly related sustainability metrics assess and evaluate how software influences sustainability beyond its direct resource usage. For example, software designed for intelligent resource management, such as IoT-based energy monitoring systems, can optimise energy consumption in industries, homes, and transportation. However, predicting these long-term effects is complex, as efficiency improvements may sometimes lead to increased overall consumption.

SSE is about reducing environmental harm and creating socially and economically responsible systems. It emphasises the balance between performance, cost-effectiveness, and environmental protection, ensuring that technological progress contributes positively to society without compromising future needs.

4. Engineering Methods For a Sustainable SDLC

Integrating sustainability into the Software Development Lifecycle (SDLC) requires awareness and practical methodologies that enable efficient resource management, waste reduction, and software efficiency. Traditional SDLC models prioritise functionality, cost, and time-to-market but do not always consider environmental and social impacts [5, 15]. Various engineering tools and frameworks have been adapted to bridge this gap and promote sustainability throughout development.

This section explores key methodologies and methods that support sustainable software development. Lean Software Development and Value Stream Mapping help eliminate inefficiencies, while Agile frameworks, such as Scrum, improve adaptability and reduce resource waste. Meanwhile, Life Cycle Assessment evaluates software's environmental impact from creation to disposal.

Integrating the right tools at each stage is essential to improving the sustainability of an SDLC. This study examines methodologies that align software development with business requirements and environmental considerations, leading to more resource-efficient and sustainable solutions. One such approach is Lean Software Development, which focuses on reducing waste, optimising workflows, and improving overall efficiency. The following subsection explores how LSD principles can be applied to create high-performing and environmentally responsible software.

4.1. Lean Software Development

Lean Software Development (LSD) is a method adapted from Lean principles initially developed in manufacturing, particularly by Toyota. The primary goal is to minimise waste and maximise value in software development, which aligns with sustainable practices by reducing unnecessary resource usage and fo-

cusing on efficiency [17].

LSD is also known by its seven principles [17, 18]. It begins by optimising the whole, ensuring synergy between its parts, which is crucial for overall success. It emphasises understanding and addressing customer problems to deliver meaningful solutions. Workers are energised by fostering creativity and inspiration, creating an environment that promotes innovation. LSD reduces friction by avoiding mistakes such as building the wrong product incorrectly or adhering to inefficient batch-and-queue processes. Continuous learning is central, enabling teams to adapt to future challenges. Increasing flow through a steady, value-driven workflow ensures efficiency while building quality into the process and promptly eliminating defects. Finally, LSD commits to continuous improvement, keeping pace with a changing world by focusing on incremental changes and applying the scientific method to refine processes and outcomes.

LSD promotes sustainability by enhancing resource efficiency, fostering flexibility, and promoting value-driven practices. By eliminating waste, LSD reduces unnecessary energy, time, and material usage, aligning with environmental goals [18]. Its iterative approach encourages adaptability, enabling organisations to respond quickly to market changes and avoid overproduction, a critical aspect of sustainable practices. Furthermore, by focusing on delivering customer value, LSD ensures that resources are allocated to meaningful outcomes, minimising the production of features or solutions that are not truly needed.

4.2. Value Stream Mapping

Value Stream Mapping (VSM) is a crucial Lean tool that visualises, analyses, and optimises the flow of materials and information throughout a process. It helps identify value-added and non-value-added activities, streamlining operations, reducing waste, and enhancing efficiency. In software development, VSM is particularly useful for understanding and improving processes such as the SDLC.

VSM begins by creating a "current state map," which outlines the existing flow of tasks and highlights bottlenecks, delays, and inefficiencies. For each step, the time spent on value-added versus non-value-added activities is calculated. Using this insight, teams can propose changes and develop a "future state map" incorporating improvements such as automation, process simplification, and enhanced communication [19].

In software development, VSM is critical in promoting sustainability by identifying inefficiencies and optimising resource usage throughout the development process. By highlighting non-value-added activities such as excessive waiting times, redundant tasks, or misaligned workflows, VSM helps teams reduce energy, time, and material waste. This streamlined approach minimises the environmental impact while maintaining high productivity. Furthermore, VSM encourages iterative improvements and agile responses to changing

requirements, reducing the need for extensive rework or discarded efforts. Aligning development processes with lean principles enhances operational efficiency and supports long-term sustainability goals in the software industry.

4.3. Kanban

Kanban is a visual workflow management tool used in the SDLC to optimise processes and enhance sustainability. It helps teams visualise their tasks, limit Work In Progress (WIP), and manage flow, ensuring efficient use of resources and reducing waste [17]. The Kanban board, a core component, provides a clear view of the development process, as illustrated in Figure 5, which shows a Kanban board used in SDLC. This enables teams to identify bottlenecks and prioritise tasks effectively.

Implementing WIP limits helps prevent bottlenecks, improve workflow, and prevent teams from doing too much work simultaneously, reducing stress and inefficiencies. This focus on completing tasks before starting new ones minimises energy and resource use, aligning with sustainable practices. Furthermore, the transparency provided by Kanban encourages better collaboration and continuous improvement.

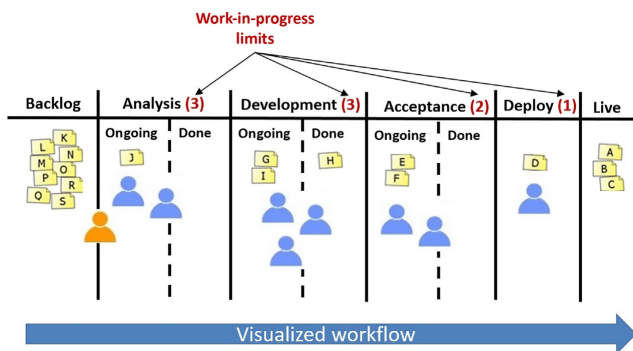


Figure 5: Example of a Kanban board for SDLC [17]

For sustainability, Kanban promotes a lean approach by avoiding overproduction, reducing defects, and optimising resource allocation. It ensures the development process remains streamlined, environmentally friendly, and aligned with customer value [17, 20].

4.4. Scrum

Scrum is an Agile framework designed to manage and develop complex software systems, emphasising collaboration, adaptability, and iterative progress. Unlike traditional software development models, which follow a rigid, linear approach, Scrum operates in short, structured sprint cycles, typically lasting between one and four weeks. Each sprint yields a functional software increment, enabling teams to continuously refine their work based on feedback and evolving requirements [17]. This flexibility makes Scrum particularly effective in dynamic environments, such as sustainability-focused

software projects, where priorities may shift.

A key aspect of Scrum's contribution to sustainability lies in its focus on resource efficiency and waste reduction. Through practices such as sprint planning and retrospectives, teams continuously assess their workflows, identifying bottlenecks and eliminating inefficiencies. This structured approach ensures that unnecessary rework is minimised and that human and computational development resources are used optimally [21]. Additionally, Scrum fosters transparency through daily stand-up meetings and backlog refinement, ensuring that sustainability-oriented features, such as energy-efficient algorithms or resource-aware system designs, are consistently prioritised.

Scrum's adaptability helps reduce software obsolescence and technical debt. Developing teams can proactively address long-term environmental and economic impacts by incorporating sustainability criteria into backlog prioritisation. For example, energy-efficient coding practices and modular architectures can be integrated into the development roadmap, ensuring that software remains scalable and maintainable over time [17, 21]. This focus on long-term viability aligns with broader sustainability goals by extending the life cycle of software products and reducing the environmental footprint associated with frequent system updates.

Scrum is more than just a project management framework; it is a tool that supports sustainable software development. Its iterative nature, focus on efficiency, and adaptability make it a practical approach for reducing waste, optimising resource usage, and fostering long-term software sustainability. Organisations can align their software engineering practices with environmental and economic sustainability objectives by embedding sustainability considerations into sprint planning and backlog management.

4.5. Life cycle assessment

Life Cycle Assessment (LCA) is a systematic method for evaluating a product or process throughout its life cycle. Applying LCA helps assess the software product and the development processes involved in software development. This approach considers all stages of the software life cycle, including development, usage, and end-of-life, and examines their direct and indirect environmental effects [22, 23].

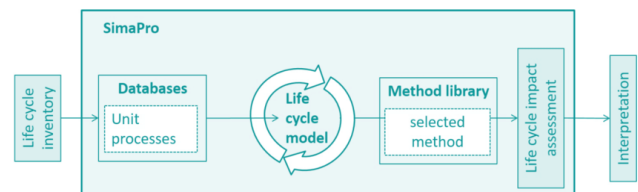


Figure 6: Use of Simapro in LCA [24]

Figure 6 [24] demonstrates how tools like Simapro facilitate LCA by using data from unit processes stored in databases. These data are fed into a life cycle

model, applying a chosen assessment method to evaluate the environmental impact. The output is analysed to understand the effects of the life cycle and interpret the results, facilitating more informed and sustainable decision-making. By adapting such tools for software engineering, we can categorise environmental impacts, standardise their effects, and prioritise them based on their importance. This ensures that sustainability becomes a key factor in design and process decisions, aligning software development with broader ecological goals [22].

Each tool plays a distinct role in improving sustainability throughout the SDLC. As shown in Table 1, these methodologies align with different SDLC phases, ensuring that sustainability considerations are embedded at every stage. LSD is applied during the Analysis, Design, and Implementation phases, helping teams eliminate unnecessary development steps and focus on efficient resource use. VSM spans Analysis, Design, Implementation, and Testing, enabling organisations to identify inefficiencies in software processes and optimise workflows to reduce excessive energy consumption. Scrum is primarily utilised in Implementation, Testing, and Deployment, ensuring that software is developed incrementally while minimising rework and unnecessary computational processing. Kanban is particularly effective in Implementation, Testing, and Maintenance, as it helps regulate the development pace by limiting work-in-progress, preventing bottlenecks, and optimising the use of computational resources. Ultimately, LCA is applicable across all phases of the SDLC, assessing the software’s environmental footprint and providing a structured framework for continuous improvement.

Table 1 links these tools to relevant sustainability metrics, highlighting their direct contributions to energy efficiency, carbon footprint reduction, and resource optimisation. Energy consumption is a crucial metric in software sustainability, as inefficient processes lead to excessive power usage, particularly in cloud computing and data centre operations. LSD, Scrum, and Kanban contribute to reducing energy consumption by promoting streamlined development cycles, eliminating redundant tasks, and ensuring that computational resources are utilised only when necessary.

Carbon footprint reduction is another essential metric, reflecting the total greenhouse gas emissions associated with software production, execution, and maintenance. LCA plays a vital role in this area, offering a detailed evaluation of the environmental impact across the software life cycle. By integrating LCA findings, teams can make informed decisions about code efficiency, infrastructure choices, and long-term sustainability goals.

Lead time and process efficiency are critical indicators in sustainable software engineering, as prolonged development cycles often result in wasted resources, higher energy consumption, and increased operational costs. VSM, Scrum, and Kanban improve these metrics by identifying workflow inefficiencies and optimising soft-

ware delivery pipelines.

These methodologies ensure software evolves adaptively and sustainably through iterative development and continuous feedback loops. Additionally, resource efficiency is key to reducing waste in software engineering. This metric encompasses CPU usage, memory allocation, and storage efficiency, directly impacting a system’s sustainability. Kanban, VSM, and LSD contribute by preventing over-processing and optimising workloads, ensuring that only necessary computational resources are used. Scrum’s sprint-based approach further ensures that software features are developed in manageable increments, preventing excessive memory allocation and improving overall system performance. Software teams can develop systems that balance performance, sustainability, and adaptability by applying these methodologies and tracking the corresponding sustainability metrics. These tools ensure that technological advancements align with business objectives and environmental responsibility, making sustainability a core element of the SDLC rather than an afterthought.

Method	SDLC Phase	Sustainability Metrics	Purpose in Sustainability	Reference
Lean Software Development (LSD)	Analysis, Design, Implementation	Resource efficiency, CO2 reduction, waste minimization	Reduces unnecessary features and inefficient workflows to optimize software performance and sustainability.	[17, 19, 25], [18]
Value Stream Mapping (VSM)	Analysis, Design, Implementation, Testing	Process efficiency, lead time reduction, and reduced rework	Identifies inefficiencies in workflows to minimise waste and optimise energy usage.	[19, 26, 27], [28], [29] [30]
Scrum	Design, Implementation, Testing, Deployment	Team velocity, reduction of wasted time, and client satisfaction	Ensures adaptive and efficient development cycles, reducing wasted effort and computing resources.	[3, 20, 21, 31, 32]
Kanban	Implementation, Testing, Deployment, Maintenance	Flow efficiency, work-in-progress reduction, and operational stability	Limits work-in-progress to reduce unnecessary resource consumption and optimise workflow.	[17, 33, 34]
Life Cycle Assessment (LCA)	All Phases	Environmental impact, energy efficiency, carbon footprint	Evaluate software sustainability across all stages of the SDLC to minimise the ecological footprint.	[23, 24, 35]

Table 1: Aligning sustainability methods and their impact with SDLC phases.

5. Discussion

This study set out to explore how sustainability principles can be effectively integrated into the SDLC by drawing on established methods from industrial engineering. The objective was to propose an actionable approach to SSE that combines practical tools, measurable sustainability metrics, and process alignment across different phases of software development.

To this end, the research identified and adapted several engineering methods and positioned them within the SDLC according to their functional strengths. These tools were selected based on their ability to support sustainability through waste reduction, process efficiency, and iterative improvement. In parallel, a set of sustainability metrics was proposed that can be used to evaluate the sustainability of software processes in concrete terms.

Figure 7 synthesizes this approach by mapping the selected engineering methods and metrics onto the traditional SDLC phases. Each technique is positioned where it can offer the most benefit in terms of sustainable outcomes. For example, Lean and Value Stream Mapping are applied early in the lifecycle to reduce re-

source use in analysis and design stages. Agile practices like Scrum and Kanban are aligned with implementation, testing, and maintenance phases to promote adaptability and incremental progress. LCA spans the entire lifecycle, supporting continuous assessment and improvement. This figure illustrates the core proposition of this study: that integrating cross-disciplinary tools into software development enables structured, sustainable workflows.

To further consolidate the study’s insights, Figure 8 presents a visual mind map summarizing the core themes and relationships discussed throughout the paper. It links foundational sustainability concepts, such as the environmental footprint of ICT and CO2 emissions, to SDLC phases, engineering tools, and sustainability pillars (environmental, social, economic, and technical). By organizing these elements into a single reference, the mind map helps communicate the broader vision of embedding sustainability into software engineering.

While these contributions present a framework for operational sustainability in software development, the study also has limitations. From a methodological perspective, the method-phase associations proposed here are based on literature insights and theoretical reason-

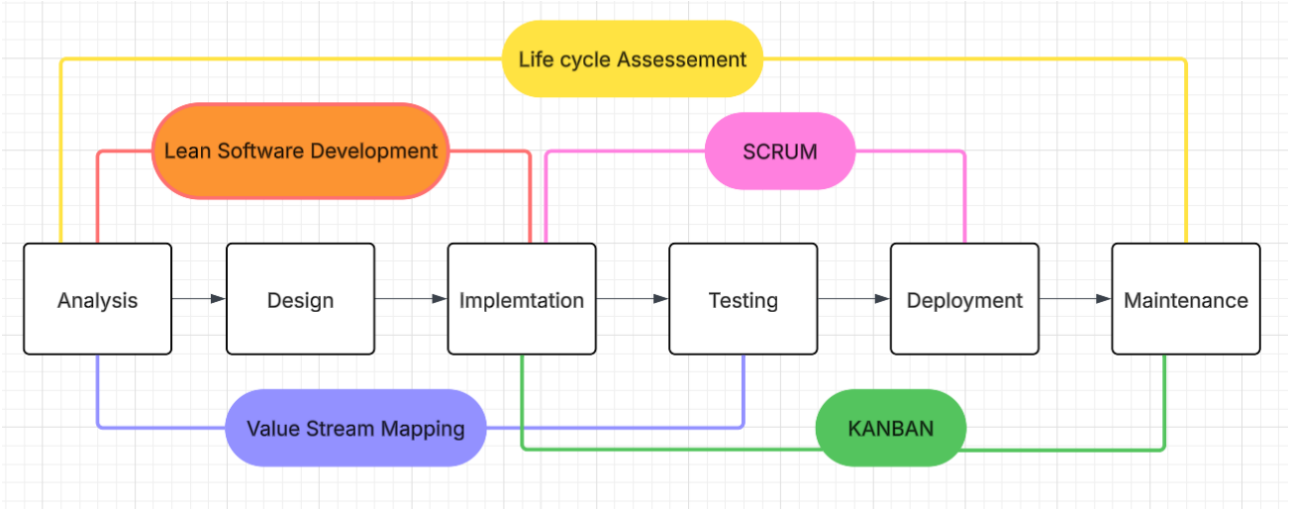


Figure 7: Mapping Engineering Methods to the SDLC Phases (created by the author)

ing rather than empirical testing. The visual models (Figures 7 and 8) are intended as exploratory frameworks and have not yet been validated in industrial settings. Additionally, the study focused on a select group of methods and metrics, other potentially relevant methods were not explored due to scope constraints.

Beyond the conceptual nature of the work, several implementation challenges remain. One of the most significant barriers to SSE adoption is the lack of standardised frameworks and widely accepted sustainability metrics [3, 36]. Although models like GREENSOFT provide structured guidance, they are still underutilised in practice due to limited awareness, resistance to process change, and the absence of regulatory incentives [11]. Sustainability also tends to be sidelined in favor of time-to-market and cost objectives. Moreover, tools like LCA are difficult to apply consistently to software systems due to their intangible and evolving nature [24].

A further challenge lies in cultural and educational readiness. Sustainability is still rarely integrated into the core values of development teams, and training programs addressing these issues remain limited [11, 17]. Without awareness and practical knowledge, efforts often remain isolated or symbolic rather than embedded within structured engineering practices.

Looking ahead, future research could focus on empirically validating the proposed framework through case studies, tool implementation, and collaboration with industry partners. There is also value in expanding the toolset to include other industrial engineering methods that may support sustainability in specific software contexts. Integrating sustainability metrics directly into software development environments could also improve usability and adoption. In the long term, including sustainability topics in university curricula and professional training will be essential for fostering responsible engineering culture.

In summary, this discussion highlights how industrial engineering methods can enhance the practice of sus-

tainable software development by offering structure, measurability, and cross-disciplinary insight. At the same time, it emphasizes the need for technical innovation and organizational awareness to fully realize the potential of Sustainable Software Engineering.

6. Conclusions

Sustainable Software Engineering is becoming increasingly relevant as the demand for digital systems grows alongside environmental concerns. This paper has highlighted the dual role of IT in both contributing to and reducing CO2 emissions, reinforcing the need for sustainable strategies in software development. By establishing clear definitions and emphasising the importance of the Software Development Lifecycle, the study provides a foundation for understanding how sustainability can be integrated into software projects.

The GREENSOFT Model and related methods such as LSD, Scrum, Kanban, VSM, and LCA offer practical pathways for improving sustainability at each development stage, they help optimize workflows, reduce energy and resource consumption, and support long-term software usability. The proposed visual mapping supports a clearer understanding of where and how sustainability practices can be applied.

To give more context, we emphasise the value of integrating methods from industrial engineering into the software development industry. Industrial engineering provides a mature set of tools for process optimisation, resource efficiency, and quality improvement, which are principles that align naturally with sustainability objectives. Otherwise, SSE brings environmental responsibility into industrial workflows, encouraging both domains to move toward greener innovation. This interdisciplinary exchange reinforces the practical relevance of SSE in industrial settings and opens up new directions for collaborative development and applied

research.

Finally, this research paves the way for practical application. In my final-year project, I plan to apply selected SSE methods, such as Lean practices and sustainability metrics, to analyse and optimise the software lifecycle. This step will help validate the concepts discussed and contribute to a deeper understanding of sustainability in software engineering through hands-on practice.

References

- [1] Eric Masanet, Arman Shehabi, Nuoa Lei, Sarah Smith, and Jonathan Koomey. Recalibrating global data center energy-use estimates. *Science*, 367(6481):984–986, 2020.
- [2] Anders SG Andrae and Tomas Edler. On global electricity usage of communication technology: trends to 2030. *Challenges*, 6(1):117–157, 2015.
- [3] Stefan Naumann, Markus Dick, Eva Kern, and Timo Johann. The greensoft model: A reference model for green and sustainable software and its engineering. *Sustainable Computing: Informatics and Systems*, 1(4):294–304, 2011.
- [4] Eva Kern, Stefan Naumann, and Markus Dick. Processes for green and sustainable software engineering. *Green in Software Engineering*, pages 61–81, 2015.
- [5] Colin C Venters, Rafael Capilla, Stefanie Betz, Birgit Penzenstadler, Tom Crick, Steve Crouch, Elisa Yumi Nakagawa, Christoph Becker, and Carlos Carrillo. Software sustainability: Research and practice from a software architecture viewpoint. *Journal of Systems and Software*, 138:174–188, 2018.
- [6] Lotfi Belkhir and Ahmed Elmeligi. Assessing ict global emissions footprint: Trends to 2040 & recommendations. *Journal of cleaner production*, 177:448–463, 2018.
- [7] Erol Gelenbe and Yves Caseau. The impact of information technology on energy consumption and carbon emissions. *ubiquity*, 2015(June):1–15, 2015.
- [8] Mou Mahmood, Prangon Chowdhury, Rahbaar Yeassin, Mahmudul Hasan, Tanvir Ahmad, and Nahid-Ur-Rahman Chowdhury. Impacts of digitalization on smart grids, renewable energy, and demand response: An updated review of current applications. *Energy Conversion and Management: X*, page 100790, 2024.
- [9] IRENA. *Renewable Energy Statistics 2019*. The International Renewable Energy Agency, Abu Dhabi, 2019.
- [10] Nina Wolfram, Patricia Lago, and Francesco Osborne. Sustainability in software engineering. *2017 Sustainable Internet and ICT for Sustainability (SustainIT)*, pages 1–7, 2017.
- [11] Raisian Komeil, Yahaya Jamaiah, and Deraman Aziz. Current challenges and conceptual model of green and sustainable software engineering. 2016.
- [12] Jamaiah Yahaya, Siti Ibrahim, Komeil Raisian, and Aziz Deraman. Green software process based on sustainability dimensions: The empirical investigation. In *Proceedings of the 1st International Conference on Informatics, Engineering, Science and Technology, INCITEST 2019, 18 July 2019, Bandung, Indonesia*, 2019.
- [13] Marco Kuhrmann, Philipp Diebold, Jürgen Münch, Paolo Tell, Vahid Garousi, Michael Felderer, Kitija Trektare, Fergal McCaffery, Oliver Linssen, Eckhart Hanser, et al. Hybrid software and system development in practice: waterfall, scrum, and beyond. In *Proceedings of the 2017 international conference on software and system process*, pages 30–39, 2017.
- [14] Mikael Octavinus Chan, Setiadi Yazid, et al. A novel framework for information security during the sdlc implementation stage: A systematic literature review. *Jurnal RESTI (Rekayasa Sistem dan Teknologi Informasi)*, 8(1):88–99, 2024.
- [15] Jide ET Akinsola, Afolakemi S Ogunbanwo, Olatunji J Okesola, Isaac J Odun-Ayo, Florence D Ayegbusi, and Ayodele A Adebisi. Comparative analysis of software development life cycle models (sdlc). In *Intelligent Algorithms in Software Engineering: Proceedings of the 9th Computer Science On-line Conference 2020, Volume 1 9*, pages 310–322. Springer, 2020.
- [16] San Murugesan. Harnessing green it: Principles and practices. *IT professional*, 10(1):24–33, 2008.
- [17] Pilar Rodríguez, Mika Mäntylä, Markku Oivo, Lucy Ellen Lwakatare, Pertti Seppänen, and Pasi Kuvaja. Advances in using agile and lean processes for software development. In *Advances in computers*, volume 113, pages 135–224. Elsevier, 2019.
- [18] LOFB Dalla. Lean software development practices and principles in terms of observations and evolution methods to increase work environment productivity. 2020.
- [19] Bong Keun Jeong, Soo-Yeon Ji, and Dong Hyun Jeong. Lean it with value stream mapping analysis: A case study in software development life cycle process. *Information Resources Management Journal (IRMJ)*, 35(1):1–18, 2022.
- [20] Nala Freedrikson Arifin, Betty Purwandari, and Farisya Setiadi. Evaluation and recommendation

- for scrum implementation improvement with hybrid scrum maturity model: a case study of a new telco product. In *2020 International Conference on Informatics, Multimedia, Cyber and Information System (ICIMCIS)*, pages 178–183. IEEE, 2020.
- [21] RMDH LAVANYA. Green scrum model: Implementation of scrum in green and sustainable software engineering. *International Research Journal of Engineering and Technology*, pages 1583–1587, 2020.
- [22] Víctor Camañes, Rafael Tobajas, and Angel Fernandez. Methodology of eco-design and software development for sustainable product design. *Sustainability*, 16(7):2626, 2024.
- [23] Matthias Galster. Life-cycle assessment in software engineering. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering*, 2010.
- [24] J Almeida, ML Pellec, and J Bengtsson. Reusable coffee cups life cycle assessment and benchmark. *Edge: Joondalup, Australia*, 2018.
- [25] Kai Petersen and Claes Wohlin. Measuring the flow in lean software development. *Software: Practice and experience*, 41(9):975–996, 2011.
- [26] Shekhar Tankhiwale and Sachin Saraf. Value stream mapping (vsm) led approach for waste and time to market reduction in software product development process. *Telecom Business Review*, 13(1):27, 2020.
- [27] Shahid Mujtaba, Robert Feldt, and Kai Petersen. Waste and lead time reduction in a software product customization process with value stream maps. In *2010 21st Australian software engineering conference*, pages 139–148. IEEE, 2010.
- [28] Nauman Bin Ali, Kai Petersen, and Kurt Schneider. Flow-assisted value stream mapping in the early phases of large-scale software development. *Journal of Systems and Software*, 111:213–227, 2016.
- [29] Nauman Bin Ali, Kai Petersen, and Breno Bernard Nicolau De França. Evaluation of simulation-assisted value stream mapping for software product development: Two industrial cases. *Information and software technology*, 68:45–61, 2015.
- [30] M VenkataSwamy, Hrushikesh Mohanty, and Srini Ramaswamy. Vsm: A visual tool for the design and deployment of security requirements. In *2011 Annual IEEE India Conference*, pages 1–6. IEEE, 2011.
- [31] RMDH Lavanya. Green scrum model: Implementation of scrum in green and sustainable software engineering. *International Research Journal of Engineering and Technology*, pages 1583–1587, 2020.
- [32] Hatf Shamshiri, Ashok Tripathi, Shola Oyediji, and Jari Porras. Exploring the experiences of experts: Sustainability in agile software development—insights from the finnish software industry. *arXiv preprint arXiv:2407.06978*, 2024.
- [33] Muhammad Ovais Ahmad, Denis Dennehy, Kieran Conboy, and Markku Oivo. Kanban in software engineering: A systematic mapping study. *Journal of Systems and Software*, 137:96–113, 2018.
- [34] Erika Corona, Filippo Eros Pani, et al. A review of lean-kanban approaches in the software development. *WSEAS transactions on information science and applications*, 10(1):1–13, 2013.
- [35] Ivan T Herrmann and Andreas Moltesen. Does it matter which life cycle assessment (lca) tool you choose?—a comparative assessment of simapro and gabi. *Journal of Cleaner Production*, 86:163–169, 2015.
- [36] Coral Calero, Félix O García, Gabriel Alberto García-Mireles, M Moraga, and Aurora Vizcaino. Addressing sustainability-in software challenges. *arXiv preprint arXiv:2406.07380*, 2024.

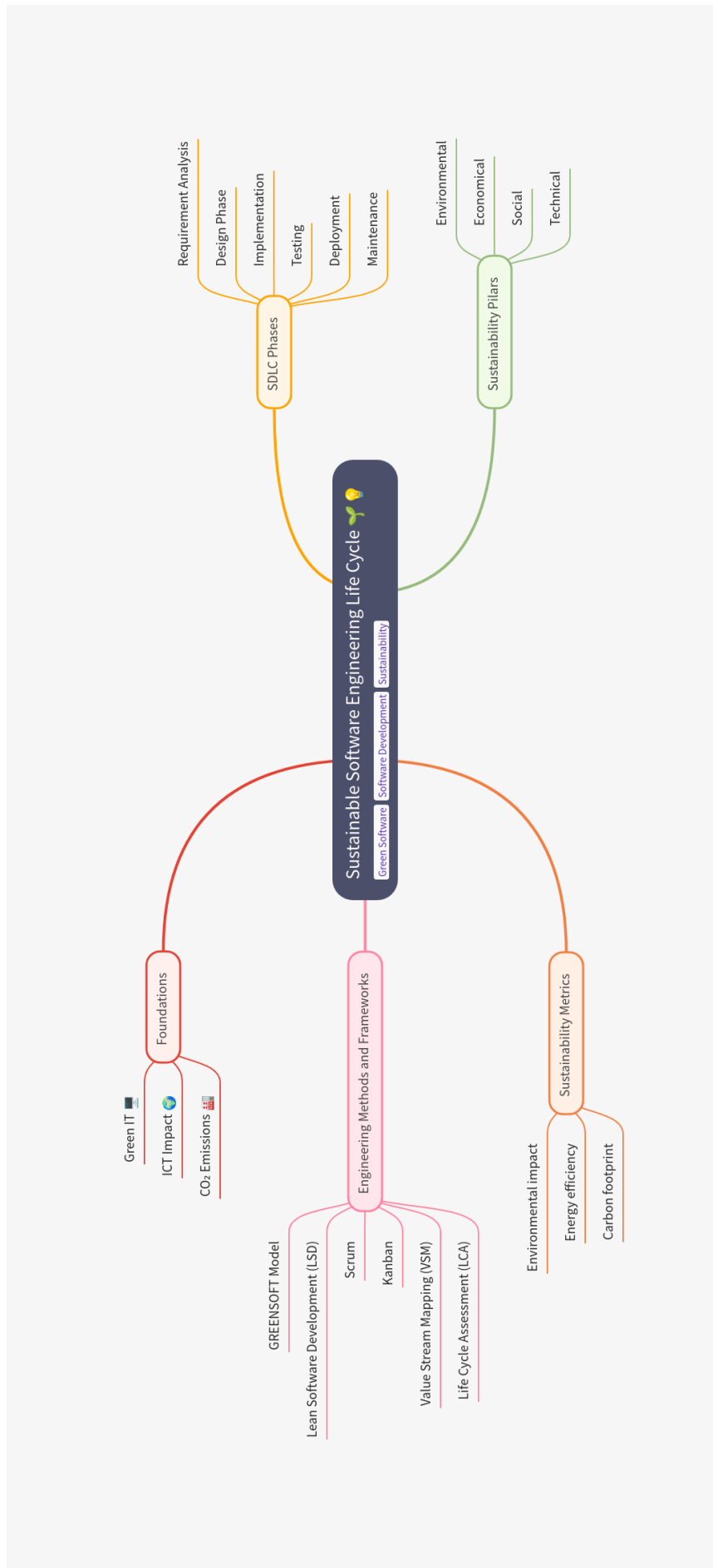


Figure 8: Sustainable Software Engineering Life Cycle Mind Map (created by the author)