A Systems Thinking Approach to Improve Sustainability in Software Engineering—A Grounded Capability Maturity Framework

Gopalakrishnan Sriraman * and Shriram Raghunathan

School of Computing Science and Engineering, VIT Bhopal University, Sehore 466114, India; shriram.r@vitbhopal.ac.in
* Correspondence: gopalakrishnan.sriraman2019@vitbhopal.ac.in

Abstract: Sustainability has become a critical issue for the software industry as the environmental impact of software development and use increases. To address this issue, organizations need a framework for developing and accessing sustainable software practices. In this study, we reviewed the existing literature, models, and practices in this domain as well as carried out surveys and interviews to understand the reality, practices, and challenges on the ground. We identified a set of research questions to discover why sustainability is important, what actions can be taken to improve it, and how and when they can be implemented. This study highlighted the limitations of existing models and the need to address the research gaps. Existing research is limited by a narrow focus on specific domains such as the environment and energy or a specific phase of software development. We aim to address these through the proposed comprehensive software sustainability capability framework (SSCF), which provides a “ready reckoner”, allowing any organization to assess their current software sustainability and the capabilities and metrics they could focus on to improve sustainable software maturity. This paper also provides detailed assessment criteria, metrics, and a roadmap that can be used by any software organization to enhance their sustainability.

Keywords: sustainable software; software engineering; capability framework; DevOps practices; green IT

1. Introduction

Software sustainability is a concept that focuses on the long-term viability of software systems, products, applications, and data centers. It has multiple dimensions including human, financial, and energy resources [1]. The human dimension includes engagement, psychological health, happiness, and health [2], whereas the economic dimension focuses on the total cost of ownership, development, and maintenance costs. The utilization of resources such as power and materials and their impact on the environment (e.g., their ability to be recycled and biodegradability) is also a critical dimension [3]. Green software engineering (GSE) [4] focuses on environmental sustainability, leveraging eco-friendly ways of developing systems to reduce resource utilization, thus scaling down the adverse impact on society [5]. Many researchers have focused on the energy efficiency of software and systems, operating system design, and energy efficient processes [6,7]. Deploying green software engineering practices reduces the impact of software and systems on the ecology [8,9]. Modern technology such as virtualization, cloud environment optimization, optimal data structures, and algorithms plays a vital role [10,11]. Agile and DevOps practices play an integral role in driving sustainable software development [12]. The application of Agile principles and technical Agile practices results in significant energy efficiency [13,14]. Being customer-centric, having frequent feedback cycles, and reducing waste are some of the key Agile practices for sustainability. Continuous exploration, limiting work in progress, delivering based on business demand, and shift-left testing and validation...
with built-in quality practices can reduce the total cost of ownership, thus improving the resource utilization efficiency [15–17]. Most current research has an overarching focus on a specific dimension of sustainability such as energy efficiency or Agile practices, which leads us to believe that a clear definition of sustainable software development and guidance on sustainable software development practices are lacking and that empirical research on the effectiveness of sustainable software development practices is needed. This motivated us to conduct a systematic study to answer the following research questions:

- Why is sustainable software development a priority for an organization?
- How can organizations create a roadmap for deploying sustainable software practices?
- What practices, policies, and decisions can they carry out to improve sustainability?
- When can an organization say they have reached their desired level of sustainability?

These questions have the potential to allow researchers to explore key areas and build practical approaches that organizations can implement to realize benefits. We applied a systems thinking approach in this study, which involved looking at the system rather than focusing on individual parts in isolation. We relied on the available literature and industry reports to understand the software development process as a system that includes various components such as developers, customers, and stakeholders. Understanding the system makes it possible to identify areas in which sustainability can be improved. Furthermore, we contacted practitioners and established an interactive feedback loop to gain industry insights and validate the study and assumptions. By combining the academic and industrial expertise of multiple experts, we were able to develop a capability model.

Existing models focus on specific phases of the software life cycle or a dimension of sustainability. We addressed these limitations in our model, making it novel and valuable for researchers and industry. Our work combines academic research carried out over the years with applicable industry practices to provide an evolutionary maturity model that any organization can utilize to improve its sustainability. This framework provides an end-to-end capability roadmap for sustainable software development. Section 2 provides a detailed background to the research. Section 3 presents the research questions and methodology adopted for this study. In Section 4, the survey results and interview feedback are analyzed. Section 5 details the capability framework for sustainable software development. Section 6 sets out the conclusions and future work.

2. Related Work

Sustainable software development is an active area of research, and considerable research has been carried out regarding the basic elements of sustainable software development [18]. However, this field is rapidly evolving, and the impact of sustainability at the operational level is not well-researched [19]. The scope of the research includes the study of emerging practices, technologies, and policies. The concept of sustainable software was first introduced by Babin and Marlier in 2013 [20] as “software that meets the needs of its stakeholders without compromising the ability of future generations to meet their own needs”. Stol et al. [21] elaborated on how energy efficiency contributes to sustainable software. Brunna et al. [22] summarized various approaches to sustainable software practices and provided an overview of the challenges and gaps that still need to be addressed. According to Jain and Wullert [23], software organizations are expected to use sustainability and green software leadership as key differentiators in the market. It is expected that in the future, consumers and investors may prefer software organizations that focus on sustainability as a goal. Another well-researched dimension, in addition to energy efficiency and Agile practices, is the role of knowledge management. With better knowledge and reusability, organizations can improve their sustainability and the utilization of resources [24,25].

Adopting green practices reduces the carbon footprint [26]. There is also a need to balance the effort required to adopt such practices and the value they contribute to sustainability [27].

There is a need for research and the empirical study of the impact of tools and methods, and methods for measuring the impact of practices [28–30]. A systematic review of
sustainability research in software engineering [31] identified and analyzed 106 studies published between 2007 and 2017. The study concluded that sustainability was an emerging concern in software engineering and that there was a need for more research on the integration of sustainability into software development processes and products. We also observed that there are numerous models [32,33] related to sustainable software, which are primarily classified as software product models [34] and software development process models [35–40]. The Green Software Engineering (GSE) model [41,42] was developed by researchers at the University of Sannio in Italy and focuses on integrating environmental sustainability into software development processes. The Software Sustainability Maturity Model (SSMM) [43] was developed by the Sustainable Software Consortium and is based on the Capability Maturity Model Integration (CMMI) framework. The ISO/IEC 29110 standard [44] provides guidelines for developing and maintaining software for small and medium-sized enterprises (SMEs).

To the best of our knowledge, the current research and models lack a comprehensive approach to sustainable software development in terms of answering the above questions. Most of the research focuses on improving the economic and environmental aspects, with limited attention to the human and organizational aspects of sustainability in the industry [45]. Hence, we decided to carry out a systematic study combining both academic research and industry advancements to find answers to our questions, which will help software organizations utilize academic research to improve their sustainability. Building an enterprise capability model [46] helps to develop strategy and creates a focus on developing capability maturity.

### 3. Research Methodology

We applied the grounded theory method to obtain more reliable results and gathered data from various sources and industry reports to ensure internal validity. We collected the data, analyzed, and documented the preliminary results. As confirmatory evidence, a survey of company subject matter experts was carried out, followed by semi structured interviews. Below, Figure 1 provides an overview of the overall research methodology.

| Study definition | • Purpose of the study – what and how  
| Data analysis | • Public academic literature  
| Hypothesis | • Industry reports  
| Survey and Interviews | • Summary of analysis  
| Capability Analysis and Reporting | • Assumptions and areas to clarify  
| | • Qualitative clarifications on the analysis  
| | • Validity and reliability  
| | • Capabilities driving sustainable software  
| | • Correlation and factors

**Figure 1.** Research methodology.

Step 1—Purpose definition: The ‘what’ was to identify various capabilities and levers contributing to sustainable software delivery, whereas the ‘how’ aimed to identify the specific method or characteristic of the capability. Stakeholders and software engineering practitioners in the software system integration space and captive software development centers for organizations across various industries represent the boundary for this research study.

Step 2—Industry data analysis: Software engineering has evolved over the last two decades, and significant literature and industry reports are available in the public domain. We gathered relevant literature and industry reports from across geographies to improve
our understanding of the capabilities, characteristics, and best practices contributing to sustainable software development. We reviewed the available literature from IEEE, Springer, and ACM Transactions as they represent the major publication sources for any academic research [47] as well as industry articles. PRISMA [48] methodology was used for the literature review. A detailed PRISMA diagram is provided in Appendix B, Figure A3. A total of 96 articles and industry reports were gathered from various sources using the keywords “sustainable software development”. After removing the duplicates and performing a manual review of the abstracts and summary titles, 58 were found to be relevant. The inclusion criteria were the keywords “sustainable software development”, dated from 2000 to 2022. Only English language articles were considered. Quality assessment and further study resulted in 46 articles that formed the basis for the literature review.

Step 3—Hypothesis and understanding: The industry data analysis provided a broad perspective of capabilities and characteristics. The current understanding of capabilities across various organizations was divergent. The detailed results and findings on this understanding are documented in Section 4.

Step 4—The broad understanding achieved from the industry data analysis and the resulting hypothesis were validated using a survey and focused interviews. The survey gathered large-scale, diverse insights on the capabilities contributing to sustainable software development. It was also used to reach potential subject matter experts who were available for interviews. Interview design: We devised and documented the interview questions to validate the hypothesis in the previous steps. The questions helped us to understand the ground truths and correlate the academic and industry literature with the reality of the implementation of software development and operations. The interview was semi structured using a set of predefined questions, followed by probing conversations.

Questions used as part of the process were:

RQ1 Is sustainable software development and operations a priority for the organization?
RQ2 Is there a roadmap and thought leadership within the organization for sustainable software development?
RQ3 What are the organization’s primary capabilities for driving sustainable software?
RQ4 What industry practices were followed in the specific organization, and were they successful in realizing the planned outcome?
RQ5 What challenges do organizations face to improve sustainable software development and operations?
RQ6 Do the organizations have the proper skills and technology acumen to drive sustainable software?

We sent the survey to a large population of 834 subject matter experts via their network and industry social groups. The survey was well-received, and we received 294 responses, of which, after data checks and qualification, we narrowed them down to 152 qualified responses covering diverse stakeholders. The survey statistics are presented in Appendix A, Figures A1 and A2.

As part of the survey, we requested volunteers for the interviews. Subject matter experts who had experience in software development and operations for a minimum of five years and bachelor/master’s degrees related to software engineering and information technology were chosen for an interview. We were also limited to volunteers who could provide formal consent and make themselves available for the study. We interviewed 32 subject matter experts from across geographies, companies, and roles in software organizations. Most interviews were conducted virtually using social tools. The duration of each interview ranged from 30 to 60 min, and a few involved multiple discussions to deep dive and gather insights. Consent was obtained to use the data for the research.

Step 5—Capability analysis and reporting: All the survey data were analyzed to identify patterns and insights. Interviews were fully documented to extract all the information for analysis. The analysis was performed using grounded truth theory from the literature. All of the interview data were classified and coded according to the theme. Experts (from the 32 interviewees) were selected to confirm the results and categories.
4. Results and Discussion

Sustainable software models have gained increasing attention in recent years as software development has become a critical aspect of modern society. A sustainable software model is defined as a software development approach that considers not only the immediate needs of users, but also the long-term impacts on the environment, society, and economy.

We analyzed the existing software engineering sustainability models and the emphasis and outcomes that resulted from these models. Our findings and conclusions on the limitations of the existing models are explained below in Table 1.

Table 1. The existing sustainability models and findings.

<table>
<thead>
<tr>
<th>Model Name</th>
<th>Findings and Limitations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Green Software Engineering (GSE) model</td>
<td>A narrow focus on the environmental dimension. It does not provide guidance on how to measure the environmental impact. It does not cover software maintenance and operations and is more focused on software development aspects. Lacks empirical evidence.</td>
</tr>
<tr>
<td>Software Sustainability Maturity Model (SSMM)</td>
<td>The SSMM emphasizes the importance of considering social, environmental, and economic sustainability factors in software development and maintenance. It is considered complex and prescriptive. Designed for large organizations and lacks overall industry recognition. Focused on software development but lightweight on the general organization aspects of sustainability.</td>
</tr>
<tr>
<td>ISO/IEC 29110</td>
<td>Provides a valuable framework for software development. The limited scope focuses on software development in small organizations. It covers engineering practices but lacks a focus on sustainability. It does not provide guidelines on tools and techniques.</td>
</tr>
<tr>
<td>Sustainable Software Development Framework (SSDF)</td>
<td>Offers insights into the intersection of sustainability and Agile software development practices. It takes an empirical approach, drawing on research and case studies to explore the challenges and opportunities of integrating sustainability into software development processes. Its focus is limited to the development and software engineering process aspects of sustainability.</td>
</tr>
</tbody>
</table>

The sustainable software capability models above-mentioned have a narrow focus, either emphasizing environmental sustainability only or lacking a comprehensive approach that includes social and economic sustainability factors. This limited scope can restrict the ability of organizations to address sustainability in a holistic way. Additionally, these models may lack the flexibility needed to accommodate diverse organizational contexts, software development processes, and stakeholder requirements. This can hinder the adoption of the model and its adaptation to different organizational sizes and types. Furthermore, these models often provide high-level guidelines without sufficient practical guidance on how to implement sustainable practices effectively. The models can also be resource-intensive, requiring significant time, effort, and resources to implement and maintain. Because of the above-mentioned challenges, most models are not adopted industry-wide, and this hinders benchmarking, knowledge sharing, and industry-wide collaboration. These limitations underline the need for further research and a holistic consideration of the environmental, social, technical, and organizational aspects of sustainable software delivery. Our study of the existing models and literature review analysis, together with the data gathered from the survey and interviews, provided comprehensive answers to the research questions.

RQ1 The priority of sustainable software development and operations varied from organization to organization. Some organizations might prioritize sustainability as part
of their core values and mission. In contrast, others might view it as a secondary consideration or might not have considered it at all. Of the survey respondents, 60% confirmed that sustainability was already a priority in their organizations, whereas 14% disagreed. As sustainability becomes increasingly important to consumers and stakeholders, organizations will prioritize sustainable software development and operations as part of their overall strategy. The major reasons for prioritizing sustainable computing appear to be cost optimization, improving efficiency and quality, to retain top talent, and drive innovation.

RQ2 The existence of a roadmap and thought leadership within an organization for sustainable software development varies depending on the size of the organization, the industry, and the level of commitment to sustainability. Only 21% of respondents confirmed the availability of a sustainable software development roadmap in their organization, and the majority (52%) were neutral.

RQ3 The primary capabilities for driving sustainable software in an organization can be summarized as follows: sustainable software design, development, and operations; using data analytics to monitor and measure sustainability metrics such as energy consumption, carbon footprint, and waste generation; using the insights to drive sustainability improvements; sourcing sustainable hardware, software, and other resources for software development and operations; promoting sustainable practices among suppliers and partners; educating employees on sustainable software development and operation practices and engaging them in sustainability initiatives to foster a culture of sustainability within the organization; engaging with customers, partners, and other stakeholders to promote sustainable software development and operations and drive sustainability across the value chain.

RQ4 Some of the industry best practices shared by subject matter experts included Agile and DevOps methods to promote iterative, collaborative, and efficient software development and operation practices, which can reduce waste and energy consumption. Cloud computing can enable organizations to scale their IT infrastructure efficiently, reduce hardware and energy costs, and optimize resource usage. Virtualization enables organizations to consolidate their IT infrastructure, reducing the number of physical servers and associated energy consumption. Data center optimization involves using energy-efficient hardware, cooling systems, and other technologies to reduce energy consumption and carbon emissions in data centers. Monitoring and optimizing software performance can help organizations identify opportunities to reduce energy consumption and resource usage.

RQ5 The answers to challenges can be summarized as follows. There has been a shift in the organizational culture and mindset toward long-term sustainability instead of short-term profits. Many developers and organizations lack understanding and skills in sustainable software development. Implementing sustainable practices can require significant time and financial resources, and these are limited. The absence of universally accepted standards makes it challenging to adopt sustainable practices. Few incentives exist for developers and organizations to invest in sustainable practices because of the upfront investment required. Evaluating the environmental, social, and economic sustainability of software products can be complex. Regulatory compliance: organizations may face additional challenges posed by regulatory requirements related to sustainability.

RQ6 According to the industry experts, the skill sets the industry should build for sustainable software development include sustainable software design that considers energy efficiency, reuse, and recyclability; data analytics that help monitor energy consumption and greenhouse gas emissions; cloud computing, which reduces energy consumption and emissions by providing on-demand computing resources; open-source software to promote software reuse and thus reduce environmental impact; Agile development methodologies to encourage collaboration and continuous improvement for sustainable practices; sustainable procurement to source eco-friendly
hardware and resources; user experience design to ensure software accessibility and usability for social sustainability.

In summary, sustainable software development is a priority for organizations because it aligns with the environmental and social responsibility goals, maintains long-term cost efficiency, and enhances brand reputation. To build a roadmap for deploying sustainable software practices, organizations can assess current practices, set goals, identify opportunities, develop an action plan, monitor, and measure progress, and iterate and improve. To improve sustainability, organizations can adopt energy-efficient hardware and infrastructure, emphasize clean code and efficient algorithms, implement automated testing and continuous integration, promote virtualization and cloud computing, encourage remote working, implement recycling and responsible waste management practices, evaluate, and select eco-friendly software vendors and suppliers, and educate and train employees on sustainable software development practices. An organization can consider itself to have reached its desired level of sustainability when it has achieved its predefined sustainability goals, demonstrated a consistent reduction in carbon footprint and waste generation, and embedded sustainable practices into its software development life cycle.

Overall, we also found that there needs to be more awareness and adoption of the existing models within the industry as the subject matter experts expressed ignorance and apprehension about them. These models also need to be more scalable and focus on specific dimensions, primarily software development and the environmental and energy aspects of sustainability. The above results and findings confirm that there was a need for a comprehensive capability maturity model for sustainable software development to address the limitations of the existing models above-mentioned. Hence, we propose our model.

5. Proposed Capability Maturity Model

A capability model for sustainable software development is a framework that defines the core capabilities and practices that organizations need to adopt in order to develop software sustainably. The model encompasses a range of methods and approaches that help organizations reduce the environmental impact of software development while improving economic and social sustainability.

Figure 2 shows the framework including insights from research already carried out and ground truth from industry experts via surveys and interviews.

![Software Sustainability Roadmap](image)

Figure 2. Sustainable software capability framework.
The framework is based on the capability model for sustainable software operations, which defines the core capabilities and practices that organizations need to adopt to operate software sustainably.

5.1. Levels of Maturity

The framework defines a range of maturity levels from ad hoc to leading, which organizations can use to assess their current level of maturity in adopting sustainable software operations practices, as detailed in Table 2.

Table 2. Levels of maturity—sustainable software development.

<table>
<thead>
<tr>
<th>Level</th>
<th>Maturity Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>At the ad hoc level, the organization has no formal sustainability policies or practices for software development and operations. Sustainability considerations are made ad hoc, and there needs to be a consistent approach to sustainability.</td>
</tr>
<tr>
<td>2</td>
<td>At the reactive level, the organization recognizes the importance of sustainability in software development and operations and takes some reactive measures to address sustainability issues. For example, the organization may implement energy-efficient hardware or take steps to reduce waste in data center operations.</td>
</tr>
<tr>
<td>3</td>
<td>At the proactive level, the organization takes a more systematic approach to the sustainability of software development and operations. The organization has established sustainability policies and procedures and actively seeks to integrate sustainability considerations into all aspects of software development and operations.</td>
</tr>
<tr>
<td>4</td>
<td>At the strategic level, the organization has fully integrated sustainability into its overall business strategy. The organization has a well-defined sustainability roadmap and regularly measures and reports its sustainability performance. The organization also actively engages with stakeholders to promote the sustainability of software development and operations.</td>
</tr>
<tr>
<td>5</td>
<td>Leading: At the top level, the organization is a recognized leader in sustainable software development and operation practices. The organization has established best practices and benchmarks for sustainable software development and operations and shares its knowledge and experience with other organizations. The organization also actively contributes to developing standards and guidelines for sustainable software development and operations.</td>
</tr>
</tbody>
</table>

5.2. Core Capabilities and Practices

The framework defines the core capabilities and practices that organizations must adopt to operate software sustainably and maps these to the different maturity levels. The capabilities are listed in Table 3.

Table 3. Core capabilities and practices.

<table>
<thead>
<tr>
<th>Capabilities</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sustainable design:</td>
<td>This capability focuses on designing software that minimizes the environmental impact of software and systems including security, experience, and performance.</td>
</tr>
<tr>
<td>Energy efficiency:</td>
<td>This capability focuses on developing energy-efficient software such as by reducing the computational power needed to run the software, optimizing code for energy efficiency, and using energy-efficient hardware.</td>
</tr>
<tr>
<td>Sustainable procurement:</td>
<td>This deals with decisions on procuring the right technologies such as cloud computing, use of renewable energy, and energy-efficient equipment.</td>
</tr>
</tbody>
</table>
Table 3. Cont.

<table>
<thead>
<tr>
<th>Capabilities</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Collaboration:</td>
<td>This capability focuses on collaborating with stakeholders to develop software that meets the needs of all stakeholders and on ways of working such as Agile and DevOps practices.</td>
</tr>
<tr>
<td>Continuous improvement:</td>
<td>This capability focuses on continually improving the sustainability of software development practices by monitoring the environmental impact, setting sustainability targets, and implementing procedures to reduce the environmental impact over time.</td>
</tr>
<tr>
<td>Green testing:</td>
<td>This capability focuses on developing and running tests to evaluate environmental impact such as energy consumption as a key factor in determining the software quality.</td>
</tr>
<tr>
<td>Life cycle assessment:</td>
<td>This capability focuses on assessing the environmental impact of software development throughout its life cycle, from design to disposal.</td>
</tr>
<tr>
<td>Use of open source:</td>
<td>This capability focuses on using open-source software to reduce the environmental impact of software development as open-source software can be reused, modified, and improved, reducing the need for new software development.</td>
</tr>
<tr>
<td>Social responsibility:</td>
<td>This capability focuses on considering the social impact of software development such as ensuring software accessibility and usability for all users and respecting user privacy.</td>
</tr>
</tbody>
</table>

5.3. Assessment Criteria and Metrics

As set out in Table 4, the framework defines sustainability metrics that organizations can use to measure and track their progress toward achieving sustainability goals.

Table 4. Sustainability metrics for software development and operations.

<table>
<thead>
<tr>
<th>Capability</th>
<th>Assessment Criteria and Metrics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sustainable Design</td>
<td>Maintainability index (MI): MI is a software metric that measures code maintainability based on various factors such as complexity, size, and documentation. Cyclomatic complexity: This metric calculates the number of independent paths through a program module. The lower the number of paths, the more sustainable and maintainable the software. Technical debt: Technical debt is a metaphor for the additional work required to fix issues not resolved during development. It is a measure of the long-term sustainability of software architecture. Software reusability: Software reusability measures the extent to which software components can be used in other software systems. Modularity: Modularity measures how well the software is organized into modules or components. Scalability: Scalability measures how well the software can manage increasing workloads or more extensive datasets. Leverage of cloud and big data analytics as part of the foundation design to ensure it is future proof.</td>
</tr>
<tr>
<td>Energy Efficiency</td>
<td>Power consumption: Power consumption is the most direct way to measure the energy usage of a system. This metric can determine the energy consumed by the system either as a whole or by specific software components. CPU utilization: CPU utilization measures the time the CPU takes to process data. Higher CPU utilization usually leads to higher energy consumption. Memory utilization: Memory utilization measures the amount of RAM used by an application. The higher the memory usage, the more energy the system will consume. Disk utilization: Disk utilization measures the amount of disk space used by an application. The more data an application stores on a disk, the more energy the system will consume. Network utilization: Network utilization measures the network traffic an application generates. The more data an application transfers over the network, the more energy the system will consume. Code complexity: Code complexity measures the number of instructions or lines of code in a software application. Higher code complexity can lead to higher energy consumption due to increased CPU and memory usage.</td>
</tr>
</tbody>
</table>
### Table 4. Cont.

<table>
<thead>
<tr>
<th>Capability</th>
<th>Assessment Criteria and Metrics</th>
</tr>
</thead>
</table>
| **Sustainable Procurement** | Ethical sourcing: Software suppliers should be evaluated based on their ethical sourcing practices including human rights, labor rights, and environmental impact. This can be carried out through a supplier assessment or a third-party certification such as Fair Trade.  
Product life cycle: The life cycle of software can have a significant impact on its sustainability. Metrics such as product longevity, repairability, and recyclability can be used to evaluate the sustainability of software.  
Open source: Open-source software can be a more sustainable option as it allows for collaboration and reuse of code, reducing the need for new software development and resource consumption. Metrics such as the percentage of open-source code or the use of open-source licenses can be used to evaluate the sustainability of software.  
Future-proofed technical stack: considering the latest technology developments, such as cloud, artificial intelligence, generative AI, and big data, so that the software and systems developed are relevant for the future and do not become outdated early. |
| **Collaboration**           | Team diversity: This metric measures the diversity of the team in terms of their backgrounds, experiences, and perspectives. A diverse team is more likely to produce innovative solutions and make better decisions.  
Issue resolution time: This metric measures the time it takes to resolve issues or bugs in the software. A shorter issue resolution time indicates a more efficient collaboration process.  
Code review feedback: This metric measures the feedback provided during code reviews. Constructive feedback can improve the quality of the code and promote a continuous improvement culture.  
Contribution rate: This metric measures how team members contribute to the project. A high contribution rate indicates active participation and engagement.  
User feedback: This metric measures the feedback from users of the software. Positive user feedback indicates a successful collaboration process that produces sustainable software. |
| **Continuous Improvement**  | Cycle time: This metric measures the time it takes to move a software feature from concept to production. By measuring cycle time, teams can identify bottlenecks and optimize the development process to reduce delays and improve efficiency.  
Lead time: This metric measures the time it takes to deliver a software feature to the end user. By measuring lead time, teams can identify areas where they can improve delivery speed and reduce the time it takes to obtain user feedback.  
Deployment frequency: This metric measures how frequently code changes are deployed to production. By increasing the frequency of deployments, teams can reduce the risk of introducing bugs and other issues into the software.  
Defect density: This metric measures the number of defects in the software relative to the size of the codebase. By reducing the defect density, teams can improve the overall quality of the software and reduce the time spent on bug fixing.  
Time to recovery: This measures the time it takes to recover from a production issue or outage. |
| **Green Testing**           | Automated testing: Use of automated testing tools to reduce the need for manual testing, which can save time and reduce the carbon footprint of the testing process.  
Code efficiency: This evaluates the efficiency of the code to ensure that it is optimized for resource usage and energy consumption.  
Green hosting: The selection of environmentally friendly hosting options for the software, such as cloud providers that use renewable energy sources.  
Performance optimization: This metric measures the software performance optimization to reduce its energy consumption and carbon footprint. |
| **Life Cycle Assessment**   | Energy use: This metric measures the amount of energy consumed throughout the life cycle of the software including that used during development, distribution, and use.  
Material use: This metric measures the number of materials used throughout the life cycle of the software including any physical components of the software  
Social impact: This metric measures the social impact of the software including its effect on user privacy, data security, and other social issues.  
End-of-life considerations: This metric evaluates the end-of-life considerations relating to the software including the ability to recycle or dispose of any physical components and any plans for decommissioning or retiring the software. |
Table 4. Cont.

<table>
<thead>
<tr>
<th>Capability</th>
<th>Assessment Criteria and Metrics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use of Open Source</td>
<td>Contribution diversity: This metric measures the diversity of contributions to the open-source project. By measuring the diversity of contributors, developers can ensure that the software is developed in a way that is inclusive and representative of different perspectives. Release frequency: This metric measures how frequently new software releases are made. By releasing software updates often, developers can ensure that the software remains relevant and current. Community health: This metric measures the health of the community engaged in the open-source project.</td>
</tr>
<tr>
<td>Social Responsibility</td>
<td>Design software with sustainability in mind: Developers can design energy-efficient software that uses fewer resources such as memory and storage. Ensure accessibility: Developers can ensure their software is accessible to all users including those with disabilities or limited resources. Use ethical data practices: Companies can collect, store, and use data ethically and responsibly to protect user privacy and prevent discrimination or bias.</td>
</tr>
</tbody>
</table>

By tracking and improving these metrics over time, software development teams can create more sustainable software that is better suited to the needs of users and the environment. We tried to address the gaps and challenges in the existing models by providing an end-to-end approach addressing various dimensions of software sustainability from system design and procurement through to social responsibility, in addition to energy efficiency and environmental aspects, thereby addressing the narrow bias issues of the current models. Additionally, we provided a fluid maturity reference model that organizations can adapt flexibly, avoiding rigid policies. A clear roadmap with potential metrics provides clarity and enables organizations to focus on tangible outcomes that can be measured. The maturity model is not focused on organizations of a specific size, but is simple and adaptable for organizations of any size. The framework is built on existing models, and our aim was to provide detailed checklists and an assessment framework so that organizations can jumpstart their sustainability journey.

5.4. Implications for Theory and Practice

From a theoretical perspective, the study contributes to the development of a better understanding of the concept of sustainable software development and the factors that contribute to its achievement. The study identified the limitations of the existing sustainable software capability models and proposes a new model that addresses these limitations and provides a more comprehensive and integrated approach to sustainable software development. In addition to energy efficiency and environmental aspects, sustainability research also needs to address the aspects of software design, collaboration, continuous improvement, the software life cycle, open source, and social responsibility.

From a practical perspective, our study provides guidance for software development organizations seeking to adopt sustainable software development practices. The proposed sustainable software capability maturity model can be used by organizations to assess their current sustainability practices, identify areas for improvement, and develop a roadmap for sustainable software development.

5.5. Use Cases of Model

Any software organization that wishes to improve sustainability can use our capability reference model. A typical approach will involve the following steps:

1. Create an awareness and understanding of the capabilities referred to in the maturity framework.
2. Perform a mapping assessment to validate the current practices of the organization practices against the proposed model.
3. Identify the gaps and challenges related to these capabilities.
4. Create an action plan to improve the capabilities and execute the same.
5. Repeat the above steps for a defined duration, preferably every quarter until the organization reaches the desired maturity.

5.6. Limitations of the Study

Although we used a methodological approach to perform this research study, it had some limitations. Our study was limited to journal articles and available knowledge sources. However, because we ensured that we included open access information and a variety of sources, we believe that the impact of this limitation on the results is minimal. The survey participants and interview group were limited to our network and volunteers from the LinkedIn social network. However, we had a statistically diverse set of participants from across geographies and organizations related to various software sectors. Therefore, we believe that this had a minimal impact on the results. We only considered English language documents and sources because English is widely used for software research, so we do not believe this had any impact on the research outcome.

6. Conclusions and Future Work

A lot of research has been conducted on sustainable software development. Therefore, this is an appropriate time to construct a maturity model for sustainable software development that can serve as a “ready reckoner” for organizations. In this endeavor, we carried out a ground-truth-based study of the available literature and combined it with industry-leading practices from subject matter experts. The primary deliverable of this study was our proposed capability model maturity framework for sustainable software development. This framework covers the maturity levels, core capabilities and practices, assessment criteria, an improvement roadmap, and sustainability metrics, enabling organizations to contextualize their needs and improve sustainability. This study also provides a comprehensive literature analysis and an industry perspective, together with a high-level capability maturity model, for sustainable software development. In future work, we aim to build a detailed assessment framework for the proposed model that can be used by any organization. We also aim to investigate the impact of the latest AI, blockchain, and data technologies on various capabilities in the proposed model to provide reference solutions.

Author Contributions: Conceptualization, methodology, software, validation, formal analysis, investigation, resources, data curation, writing—original draft preparation, writing—review and editing, visualization, G.S.; Supervision, S.R.; Project administration, funding acquisition, G.S. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Informed Consent Statement: Informed consent was obtained from all subjects involved in the study.

Data Availability Statement: The data presented in this study are available on request from the corresponding author. The data are not publicly available due to privacy reasons.

Acknowledgments: We acknowledge all industry subject matter experts, friends, and colleagues who contributed to the survey and interviews.

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A

Total survey direct recipients: 834.
Survey respondents: 152.
Participants were limited to software and related job profiles.
Figure A1. Survey profile.

Figure A2. Survey summary.
Appendix B

<table>
<thead>
<tr>
<th>Records identified through database search (IEEE, Springer, ACM Transactions) Year range (2000–2022) (n = 1218)</th>
<th>Additional records from other sources (n = 17)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Records after duplicates removed (n = 1115)</td>
<td>Excluded (n = 1019) (Only abstract, not focused on software, not relevant, non-English etc.)</td>
</tr>
<tr>
<td>Records screened (n = 1115)</td>
<td>Full-text article excluded (n = 50) Out of scope (n – 31) Insufficient details (n = 31) Limited rigor (n = 7)</td>
</tr>
<tr>
<td>Full-text articles assessed for eligibility (n = 96)</td>
<td>Studies finally included (n = 46)</td>
</tr>
</tbody>
</table>

Figure A3. PRISMA diagram.

References

16. Tate, K. Sustainable Software Development: An Agile Perspective; Addison-Wesley: Reading, MA, USA, 2005.


Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.