International Journal on Advanced Science Engineering Information Technology

Vol.14 (2024) No. 5
ISSN: 2088-5334

Sustainable Software Solutions: A Tool Integrating Life Cycle Analysis and ISO Quality Models

Yang Qiang [a], Noraini Che Pa [a,*], Rosli Ismail [a]

[a] *Department of Software Engineering and Information Systems, Faculty of Computer Science and Information Technology, Universiti Putra Malaysia, 43400 UPM Serdang, Selangor, Malaysia*

Corresponding author: [*]norainip@upm.edu.my

***Abstract*—Sustainability is essential in software systems in today's eco-conscious atmosphere. However, companies often overlook this, resulting in energy waste and e-waste. We implemented an automated software sustainability assessment solution to address this by fusing Life Cycle Analysis (LCA) and ISO quality model implementation capabilities. Our tool reduces environmental impact, promotes economic, technical, environmental, and social sustainability, improves resource labor efficiency and usage time (hardware life), and enables user autonomy. Developed in Python, it is a tool for assessing and evaluating software sustainability (such as performance or maintainability) that has been validated in real-world scenarios. We provide a method for assessing software maintainability and energy efficiency by combining LCA with the ISO 25010 standard. Case studies confirm that the new tool offers a comprehensive sustainability assessment method consistent with sustainable development goals. The study results show that maintainability and energy efficiency were thoroughly assessed. The accuracy and precision of the test assessment results were further confirmed, indicating that the tool is consistent across different software projects and reliable, proving its practical application. This advancement is essential for sustainable software development and provides concrete metrics and operational insights for developers and their companies. In future research, we plan to extend the tool's metric scope and improve data visualization/information customization suitable for more diverse software environments and sustainability goals. Our tool promotes eco-responsibility while raising quality and sustainability standards for all software systems.**

***Keywords*—Sustainability; life cycle analysis; ISO.**

Manuscript received 5 Mar. 2024; revised 21 Jun. 2024; accepted 9 Sep. 2024. Date of publication 31 Oct. 2024.

I. INTRODUCTION

With the rapid advancement of technology, software systems have become integral to various aspects of modern life, raising significant sustainability concerns. Traditional software development practices have primarily focused on performance, functionality, and usability, often neglecting the broader environmental, economic, and social impacts [1]. This project addresses these gaps by developing a comprehensive software sustainability assessment tool that integrates Life Cycle Assessment (LCA) with the ISO 25010 quality model. The ISO 25010 model provides a robust framework for evaluating software quality, focusing on performance efficiency, maintainability, and availability [2]. However, it does not explicitly consider environmental sustainability. This project seeks to fill this gap by incorporating LCA, which evaluates the environmental impacts of products across their entire lifecycle—from raw material extraction to end-of-life disposal [3]. By integrating LCA with ISO 25010, the project aims to provide a more holistic evaluation of software sustainability, encompassing traditional quality attributes and indicators [4]. This research uniquely contributes to the field by combining the Life Cycle Assessment (LCA) with the ISO 25010 quality model, providing a novel framework for assessing software sustainability. Unlike existing models that treat software quality and environmental impact as separate entities, our approach unifies these aspects and ensures a comprehensive assessment throughout the software lifecycle. This integration bridges the gap between software quality and sustainability and provides a practical tool that can be used in real-world scenarios to achieve sustainability goals.

The primary problem this study addresses is the need for comprehensive sustainability metrics in existing software quality models. Most current models, including ISO 25010, overlook critical sustainability metrics, limiting the scope of

their assessments and preventing software development from fully supporting broader sustainability goals [5]. Additionally, there needs to be more effective integration between LCA and software quality models, which hampers the ability to assess environmental impacts during the software development process [6]. Furthermore, standard practices often neglect the ecological sustainability of software, such as its energy consumption and e-waste generation, leading to a significant ecological footprint [7].

The objectives of this project are threefold: first, to integrate ISO and LCA principles into a comprehensive framework for evaluating software sustainability; second, to develop practical tools for assessing software sustainability that incorporate this integrated framework; and third, to empirically validate the effectiveness of these tools in real-world software development scenarios [8]. By achieving these objectives, the project aims to enhance the ability of software developers and organizations to make informed decisions that support both quality and sustainability goals [9].

This project's scope includes designing and implementing a software sustainability assessment tool that integrates LCA with the ISO 25010 quality model. This involves identifying relevant environmental indicators and incorporating them into the ISO 25010 framework, focusing mainly on maintainability and energy efficiency [2]. The tool will be developed using Python and validated through case studies to ensure practical applicability. Continuous improvement mechanisms will be established to incorporate user feedback and the latest advancements in sustainability assessment and software quality modeling [10]. This project aligns with the Sustainable Development Goals (SDGs) by promoting sustainable software practices. It aims to reduce the ecological footprint of software systems, thereby supporting broader environmental and sustainability initiatives [11].

Software sustainability has gained increasing importance as software systems' environmental, economic, and social impacts are more widely recognized. Existing literature on software sustainability, Life Cycle Assessment (LCA), the ISO 25010 quality model, related works, and selected indicators and evaluation standards provides a foundation for developing a comprehensive software sustainability assessment tool that integrates LCA with ISO 25010 [12].

Software sustainability refers to software systems' long-term environmental, economic, and social impacts throughout their lifecycle. It encompasses technical, environmental, social, and economic dimensions [13]. Technical sustainability focuses on maintainability, ensuring software can be easily modified and improved, thus reducing resource wastage. Environmental sustainability involves optimizing energy consumption and reducing carbon emissions and e-waste [14]. Social sustainability addresses user satisfaction, privacy, and ethical considerations, while economic sustainability aims at cost-effectiveness and financial viability [15]. Recent studies emphasize the need for holistic software engineering practices integrating sustainability at all lifecycle stages, from design and development to deployment and maintenance [16]. For instance, Penzenstadler et al. (2014) highlighted the importance of considering sustainability throughout the software lifecycle to minimize operational costs, improve software quality, and increase user satisfaction [1]. Albertao et al. [17] further noted that sustainable software development aligns with broader organizational goals of corporate social responsibility and environmental stewardship.

Life Cycle Assessment (LCA) is a systematic method for assessing the environmental impacts of a product, process, or service across its entire lifecycle. It includes four main stages: goal and scope definition, inventory analysis, impact assessment, and interpretation [2]. LCA provides valuable insights into the environmental performance of products and processes, making it a crucial tool for enhancing software sustainability. In software engineering, LCA helps quantify energy consumption and carbon footprint during software development and operation, enabling optimization for better environmental performance [4]. For example, Calero and Piattini [2] discussed the application of LCA in software development to assess the environmental impacts of software systems, highlighting its capability to quantify energy consumption and carbon footprint. Penzenstadler et al. [18] emphasized the significance of LCA in identifying critical stages in the software development process to mitigate environmental impacts effectively. Despite the challenges posed by the dynamic nature of software lifecycles, LCA offers significant opportunities for improving environmental impacts [19].

The ISO 25010 quality model is an international standard for assessing software quality. It defines eight key quality characteristics: functional suitability, performance efficiency, compatibility, usability, reliability, security, maintainability, and portability [20]. While ISO 25010 provides a comprehensive framework for evaluating software quality, it does not explicitly include environmental sustainability. Integrating LCA with ISO 25010 can bridge this gap, allowing for a more holistic assessment with sustainability metrics [4]. This integration enables development teams to systematically identify and address sustainability aspects, enhancing software quality and reducing environmental impacts [21]. For example, Calero et al. [4] proposed extending ISO 25010 by incorporating sustainability characteristics into the quality model, allowing for a comprehensive evaluation of the software's environmental impact. Boarim et al. [8] demonstrated the practical application of ISO 25010 in assessing Customer Relationship Management (CRM) systems, improving functional suitability and performance efficiency.

Various approaches and tools have been developed to assess software sustainability. These include the Global Sustainability Index, which simplifies sustainability reporting using mathematical algorithms, and the Sustainability Assessment Framework (SAF), which integrates sustainability-quality requirements into software projects [22]. The Global Sustainability Index addresses the heterogeneity of sustainability indicators through systematic aggregation and evaluation methods, offering valuable insights into sustainable management practices [23]. Grecu et al. [24] highlighted the tool's adaptability in handling imprecise and scarce information, making it a valuable asset for organizations striving toward sustainable practices. The SAF employs a participatory and technical action research methodology, enhancing its applicability in the software industry [25]. Workshops and surveys have identified gaps in understanding and applying sustainability principles among software developers, highlighting the need for better training

and guidelines [7]. For instance, Oyedeji et al. [26] conducted workshops with software developers to identify key sustainability concerns, finding that economic and technical dimensions are often prioritized over social sustainability. Noman et al. [7] surveyed software professionals and found significant gaps in understanding sustainability principles, emphasizing the need for comprehensive training and guidelines.

The GREENSOFT model offers a comprehensive framework for sustainable software design and development, addressing sustainability criteria across all lifecycle phases [12]. This model includes a lifecycle model for software products, sustainability metrics, and criteria, providing a cradle-to-grave approach to software lifecycle management [12]. Naumann et al. [12] described the GREENSOFT model's extensive coverage of sustainability criteria, emphasizing its role in balancing ICT's resource and energy consumption with its benefits in solving environmental issues.

This project focuses on maintainability and energy efficiency as crucial indicators for software sustainability. As defined by ISO 25010, maintainability ensures that software can be easily updated and repaired, reducing resource consumption and waste [27]. By ensuring high maintainability, organizations can extend the lifespan of their software systems, reduce technical debt, and minimize the environmental impact associated with frequent replacements or extensive rework [27]. Key metrics for maintainability include Halstead metrics, cyclomatic complexity, and lines of code [27]. Energy efficiency minimizes the environmental footprint by optimizing power consumption and operational costs [28]. Key energy efficiency indicators include CPU power, DRAM power, CPU utilization, and package temperature [29]. Intel Power Gadget and other tools provide real-time data on these metrics, allowing for comprehensive analysis and optimization of software energy consumption [24].

In summary, the literature on software sustainability, LCA, the ISO 25010 quality model, and related works highlight the need for an integrated approach to assess software sustainability comprehensively [18]. By focusing on maintainability and energy efficiency, this project aims to develop a robust tool that advances sustainable software engineering practices, aligns with the Sustainable Development Goals (SDGs), and reduces the ecological footprint of software systems [12]. The following Table I summarizes the related works。 ·

TABLE I
SUMMARY OF RELATED WORKS

Authors	Year	Title	Methodology/Tool	Key Findings	Limitations
Oyedeji et al.[24]	2021	Software sustainability: academic understanding and industry perceptions	Workshops with software developers	Identified economic and technical dimensions as primary concerns	Limited focus on social dimensions
Noman et al. [7]	2022	An exploratory study of software sustainability at early stages of software development	Survey of software professionals	Highlighted the need for better understanding of sustainability	Confusion between "green software" and "sustainable software"
Grecu et al. [1]	2020	Software Application for Organizational Sustainability Performance Assessment	Mathematical algorithms for sustainability reporting	Simplified sustainability reporting process	Accuracy issues with complex environmental data
Boarim & Da Rocha[8]	2020	Evaluating CRM Systems with ISO 25010	ISO 25010 quality evaluation	Improved functional suitability and performance efficiency	Specific to CRM systems, not generalizable to all software types
Condori-Fernandez et al. [23]	2022	An Action Research for Improving the Sustainability Assessment Framework Instruments	Participatory and technical action research	Validated SAF's applicability in identifying sustainability-quality requirements	Limited by the specific cases and software products used in the study
Naumann et al. [12]	2011	The GREENSOFT Model: A reference model for green and sustainable software and its engineering	Development of a conceptual reference model	Provides a comprehensive framework for sustainable software across all life cycle phases	Requires further empirical validation across different software domains

II. MATERIAL AND METHOD

Integrating life Cycle Assessment (LCA) with the ISO 25010 quality model is the methodology for developing a software sustainability assessment tool. This approach ensures a systematic and rigorous process by combining theoretical insights with practical implementation to achieve the research objectives. The methodology is divided into several key sections: theoretical research, integration, design, development, and evaluation.

The theoretical research involved an extensive literature review to understand key concepts in software sustainability, LCA, and ISO quality models. The review focused on identifying gaps and potential improvements in software sustainability practices. Key findings from previous studies were summarized to form the foundation for the project's framework.

The integration phase combined insights from the theoretical research to develop a cohesive framework that merges ISO 25010 standards with LCA principles. This involved mapping ISO 25010 quality characteristics to LCA stages and defining specific sustainability indicators. Expert reviews and preliminary tests ensured the integrated framework's robustness and applicability.

The integrated framework was translated into a practical tool in the design phase. The system architecture was detailed, including user interface design and component interactions. The design process also involved creating algorithms to calculate sustainability scores based on defined indicators and establishing a database schema for efficient data management using MySQL.

The development phase implemented the design using Python for its versatility and PyQt5 for a user-friendly graphical interface. The tool's backend was developed to handle data processing, while the frontend focused on ensuring responsiveness and usability. Rigorous testing, including unit and integration tests, ensured the tool's functionality and reliability.

The evaluation phase involved testing the tool with real-world software projects to validate its effectiveness. Specific evaluation standards and indicators were defined, and the tool's performance was assessed through practical application. Feedback mechanisms were established to enable continuous improvement based on user input and the latest advancements in sustainability assessment.

By following this structured methodology, the project aims to develop a robust, reliable, and comprehensive tool for assessing the sustainability of software products, contributing valuable insights into software engineering and sustainability (refer to Fig.1).

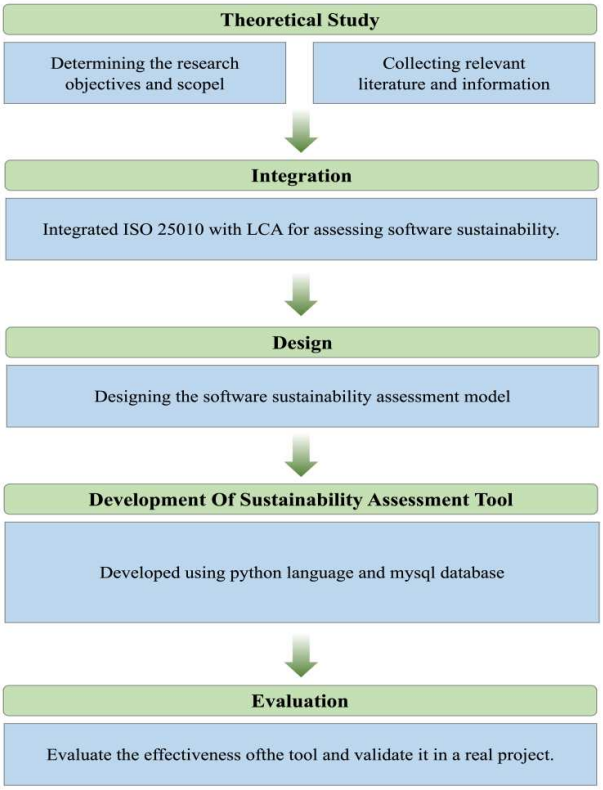

Fig. 1 Research Methodology

The analysis and design phase of a software sustainability assessment tool focuses on translating theoretical frameworks and integration methods into practical, functional tools. This phase includes use case design, framework design, data processing workflow design, and tool functionality design. My tool combines ISO to assess the sustainability of software across the life cycle. The following is a detailed analysis of the tool's phase-specific assessment methodology.

1) Requirements Analysis Phase: Although the requirements analysis phase usually involves documentation, the tool assesses the sustainability of the initial code prototypes or modules associated with the requirements. This includes assessing whether the code implements sustainability-related functionality, such as resource optimization or energy efficiency management. The tool can analyze the complexity and redundancy of the initial code structure to determine the effectiveness of requirements implementation and identify potential environmental impacts early in the development process.

2) System Design Phase: During the system design phase, the tool assesses the sustainability of design-generated code modules by analyzing maintainability and extensibility. Specifically, the tool uses static analysis to examine the code's modularity, coupling, and cohesion, which directly affect the system's long-term maintainability and resource efficiency. The tool also checks whether design patterns or optimization strategies are implemented in the code to ensure the system remains resource-efficient during future development and operation.

3) Implementation Phase: During the implementation phase, the tool assesses the sustainability of the code through static and dynamic analysis. Static analysis involves evaluating code complexity metrics (e.g., Halstead complexity, circle complexity, and lines of code) and maintainability indices, which help assess the code's long-term maintainability. Dynamic analysis involves monitoring resource usage while the code runs, including CPU utilization, memory consumption, and energy efficiency. The tool identifies high energy-consuming code segments and provides optimization suggestions to reduce energy consumption during execution.

4) Testing Phase: During the testing phase, the tool evaluates the efficiency and resource usage of the test code. It analyzes the execution time and resource consumption of test scripts and compares different testing approaches to determine the most energy-efficient testing strategy. In addition, the tool monitors real-time energy consumption during test execution to help optimize resource usage while ensuring comprehensive and efficient test coverage.

5) Deployment Phase: During the deployment phase, the tool assesses the sustainability of the deployment process by analyzing the deployment scripts or automation code. It evaluates the efficiency of the script's tasks, identifies any redundant tasks, and determines whether to use energy-efficient deployment configurations. The tool also simulates the deployment process, predicts resource consumption under different configurations, and assists in selecting the most energy-efficient deployment strategy.

6) Maintenance Phase: During maintenance, the tool assesses the system's sustainability by analyzing code changes over time. It detects if new complexity is introduced during code updates or if these changes affect the overall efficiency of the system. The tool also supports the assessment of resource usage efficiency during maintenance, e.g., whether

code optimization has reduced the system's energy consumption during updates.

7) End-of-Life: During the end-of-life phase, the tool assesses the sustainability of code execution during system decommissioning or migration to ensure that these processes do not lead to wasted resources or data leakage. The tool also provides recommendations to help the development team develop a strategy for archiving or removing code, maximizing resource recovery, and minimizing environmental impact during system decommissioning.

The use case design （as shown in Fig.2.) identifies the primary interactions between users and the software sustainability assessment tool. The leading actor, the user, can upload source code, select a programming language for analysis, run the tool, and view the generated sustainability assessment reports. The tool processes include obtaining initial results from the source code analysis, processing these results to create detailed sustainability assessment reports, and providing recommendations for improvement. The use case diagram visually represents these interactions and processes, ensuring clarity in how the tool functions and interacts with users.

The framework design（as shown in Fig.3） breaks down the core components of the tool into three major parts: data input, data processing, and data output. The data input module accepts source code from users, either through file uploads or direct code pasting. The system supports multiple programming languages to ensure broad applicability. The data processing module is the core of the assessment tool, consisting of three key processes: code extraction, maintainability indicator analysis, and energy efficiency analysis. These processes operate independently but provide necessary data for the final sustainability assessment. The data output module generates a detailed software sustainability assessment report, including scores for maintainability and energy efficiency, overall sustainability evaluation, and specific improvement suggestions.

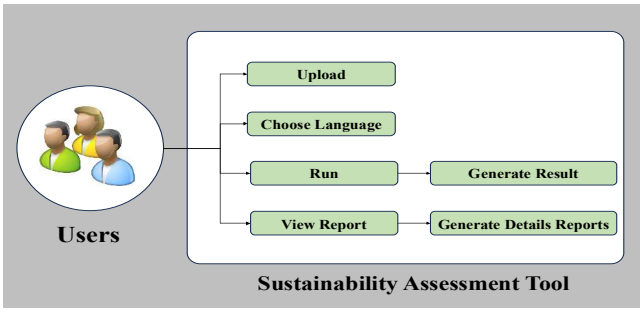

Fig. 2 Use case diagrams for software sustainability assessment tool

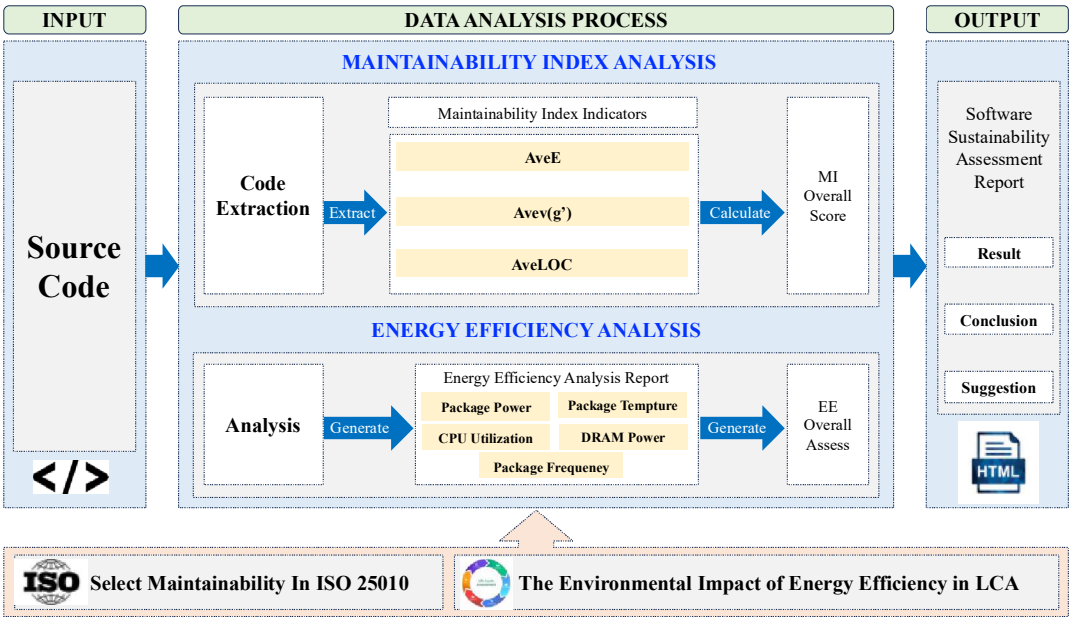

Fig. 3 Framework for software sustainability tools

The flowchart design（as shown in Fig.4.）provides a step-by-step visual representation of the tool's operation. It begins with user input of the source code, followed by parallel paths for extracting maintainability metrics and integrating tools for energy efficiency analysis. The extracted data is then consolidated and processed to generate a detailed assessment report. This flowchart ensures that each step of the process is clearly defined and logically sequenced, facilitating smooth and efficient tool operation.

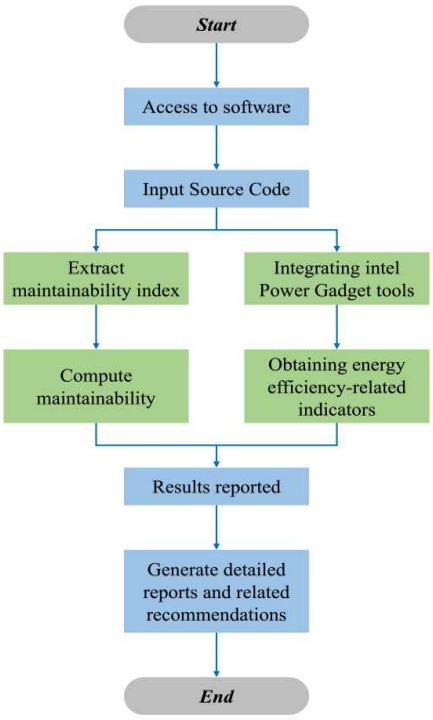

Fig. 4 Flowchart for Software Sustainability Tool

The functional design outlines the significant features and capabilities of the tool. Key features include source code input, where users can input source code in multiple programming languages through file uploads or direct pasting. Using static and dynamic analysis tools, the tool extracts maintainability and energy efficiency metrics from the source code. It calculates maintainability indices based on Halstead metrics, cyclomatic complexity, and lines of code. Additionally, it analyzes energy efficiency data, including CPU power consumption, DRAM power, CPU utilization, and package temperature. The tool then generates detailed assessment reports summarizing the analysis results and providing specific recommendations for improving software sustainability.

By integrating these features, the tool aims to provide comprehensive insights into the maintainability and energy efficiency of the input source code, helping developers optimize and improve their code effectively. This phase ensures the design is user-centric, functional, and capable of delivering valuable sustainability assessments.

The analysis and design phase establishes a clear and structured approach to developing the software sustainability assessment tool. This phase lays the groundwork for creating a robust and practical tool that supports sustainable software development practices by focusing on use case design, framework design, data processing workflows, and functional capabilities.

The Software Sustainability Assessment Tool (SSAT) implementation phase translates the design into a functional application through systematic steps, including environment setup, back-end and front-end coding, database integration, and complete testing.

First, the development environment was configured with Python (for its large number of libraries), PyQt5 (for its responsive graphical user interface (GUI)), and MySQL (for its powerful data management capabilities.) Tools such as Anaconda and Spyder facilitated development, and Intel Power Gadget was used for real-time energy efficiency data Back-end development focused on implementing the core logic and

Back-end development focused on implementing the core logic and algorithms. Functions to calculate maintainability indices were developed using Halstead metrics, cyclomatic complexity, and lines of code. Energy efficiency analysis included the integration of an Intel Power Gadget to measure CPU power, DRAM power, CPU utilization, and package temperature to provide a comprehensive software energy efficiency assessment.

Front-end development creates an intuitive interface that allows users to enter source code via file upload or direct paste, select a programming language, and specify a report output path. Key features included buttons to upload code, run the assessment and view results, ensuring a straightforward user experience.

Database integration included designing a MySQL schema for storing source code information, maintainability metrics, and energy efficiency data. Efficient data storage, retrieval, and update operations were implemented to ensure the security and accessibility of the assessment data.

Rigorous testing and debugging were also performed to ensure reliability and accuracy. Unit tests verified the functionality of individual components, while integration tests ensured seamless operation of all elements. Detailed test cases covering a wide range of scenarios ensured the tool's robustness in handling different types of source code and generating accurate assessments.

By completing the implementation phase, the project ensured the software sustainability assessment tool was functional and user-friendly, providing valuable insights into software maintainability and energy efficiency. This phase laid the groundwork for future enhancements and continuous improvements based on user feedback and advances in sustainability assessment methodologies.

III. Result and Discussion

The evaluation involved rigorous tool testing using several software projects to ensure the thoroughness and representativeness of the results. Experiments were conducted on a consistent hardware and software platform, including an Intel i7 processor and 16 GB RAM, running Windows 10 and Python 3.8 with its dependencies. The tool's maintainability and energy efficiency metrics were collected and compared against manual evaluations and other existing tools to validate its precision.

A. Case Study

Case Study 1: Small to Medium-Sized Web Application This case study evaluated a web application designed for managing online content and user interactions. The application, developed using HTML, CSS, JavaScript, and Python (Django framework), performed standard tasks such as user authentication, data management, and content creation. The maintainability analysis indicated an average Halstead Volume of 35.4, cyclomatic complexity of 4.2, lines of code at 150, and a maintainability index of 82.7, suggesting the code is well-structured and relatively easy to maintain.

Energy efficiency analysis showed a package power consumption of 25W, DRAM power of 3.8W, CPU utilization at 55%, and a package temperature within acceptable limits, highlighting areas for optimization.

Case Study 2: Large Enterprise Application The second case study focused on a large enterprise customer relationship management (CRM) application. The application was developed using Java and Spring framework, with functionalities including customer data management, sales tracking, and reporting. The maintainability analysis revealed an average Halstead Volume of 48.2, cyclomatic complexity of 6.5, lines of code at 800, and a maintainability index of 75.3, indicating higher complexity and potential areas for improvement. Energy efficiency analysis showed a package power consumption of 40W, DRAM power of 5.5W, CPU utilization at 70%, and elevated package temperatures, emphasizing the need for performance optimization.

B. Expert Survey

To evaluate the usability and effectiveness of the software sustainability assessment tool, we conducted an expert survey involving a group of professionals in software engineering and sustainability. These experts were carefully selected based on their extensive experience and knowledge in software development, sustainability practices, and quality assessment. We used the System Usability Scale (SUS) survey, a widely recognized tool for evaluating system usability. Each expert participant was asked to rate these statements based on their experience using the tool. This structured feedback provided valuable insights into how well the tool meets the needs of professionals in the industry.

The SUS consists of 10 questions (as shown in Fig.5), each rated on a 5-point Likert scale from "strongly disagree" to "strongly agree". The survey covers the ease of use, functionality, complexity, and user confidence in using the system. The significance of the SUS scale is to quantitatively assess the overall user satisfaction and usability of the system and to provide data to support the improvement and optimization of the system

Expert 1

	The System Usability Scale Standard Version	Strongly disagree				Strongly agree
		1	2	3	4	5
1	I think that I would like to use this system.	○	○	○	○	●
2	I find this system to be logical.	○	○	○	○	●
3	I thought the system was easy to use.	○	○	○	○	●
4	I think that I would need the support of a technical person to be able to use this system.	○	○	○	●	○
5	I found the various functions in the system were well integrated.	○	○	○	●	○
6	I think there is good consistency in this system.	○	○	○	○	●
7	I would imagine that most people would learn to use this system very quickly.	○	○	○	○	●
8	I found the system very easy to use.	○	○	○	○	●
9	I felt very confident using the system.	○	○	○	○	●
10	I don't needed to learn a lot of things before I could get going with this system.	○	○	○	○	●

Expert 2

	The System Usability Scale Standard Version	Strongly disagree				Strongly agree
		1	2	3	4	5
1	I think that I would like to use this system.	○	○	○	●	○
2	I find this system to be logical.	○	○	○	●	○
3	I thought the system was easy to use.	○	○	○	○	●
4	I think that I would need the support of a technical person to be able to use this system.	○	○	○	○	●
5	I found the various functions in the system were well integrated.	○	○	○	○	●
6	I think there is good consistency in this system.	○	○	○	○	●
7	I would imagine that most people would learn to use this system very quickly.	○	○	○	●	○
8	I found the system very easy to use.	○	○	○	○	●
9	I felt very confident using the system.	○	○	○	○	●
10	I don't needed to learn a lot of things before I could get going with this system.	○	○	○	●	○

Expert 3

	The System Usability Scale Standard Version	Strongly disagree				Strongly agree
		1	2	3	4	5
1	I think that I would like to use this system.	○	○	○	○	●
2	I find this system to be logical.	○	○	○	●	○
3	I thought the system was easy to use.	○	○	○	○	●
4	I think that I would need the support of a technical person to be able to use this system.	○	○	○	○	●
5	I found the various functions in the system were well integrated.	○	○	○	●	○
6	I think there is good consistency in this system.	○	○	○	●	○
7	I would imagine that most people would learn to use this system very quickly.	○	○	○	●	○
8	I found the system very easy to use.	○	○	○	○	●
9	I felt very confident using the system.	○	○	○	○	●
10	I don't needed to learn a lot of things before I could get going with this system.	○	○	○	○	●

Fig. 5 Results of the Expert System Usability Assessment

The following are the demographics of the experts involved:
Expert 1: Senior Software Engineer with over 5 years of experience in software development and sustainability.
Expert 2: Intermediate Software Engineer. Has researched in the field of sustainability
Expert 3: -Intermediate Software Engineer, specializing in various tools development for many years

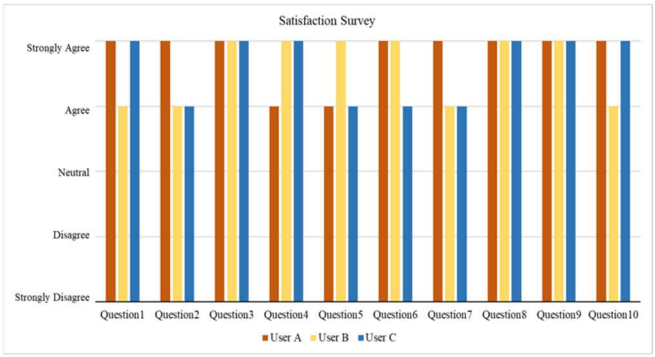

Fig. 6 Tool Evaluation Synthesis Report

In summary, the expert survey thoroughly evaluated the software sustainability assessment tool (Fig.6), highlighting its strengths in usability, accuracy, integration, and overall user satisfaction. The feedback gathered from this survey will be instrumental in further refining the tool to better serve the needs of software professionals and contribute to more sustainable software development practices.

C. Manual Measurement Comparison

We conducted a manual measurement comparison to highlight the software sustainability assessment tool's advantages. We divided the participants into two groups: one used the tool, and the other performed manual measurements. Each group consisted of three experts evaluating the same software projects (Case Study 2: Large Enterprise Application), and the results were compared.

Group 1: Tool-Assisted Assessment

Participants: Expert 1, Expert 2, Expert 3.

Group 2: Manual Assessment

Participants: Expert 4, Expert 5, Expert 6.

Tasks for Manual Assessment: For maintainability evaluation, participants from Group 2 manually calculated the Halstead metric, cyclomatic complexity, and lines of code (LOC). Then, the maintainability index was calculated using a formula. For energy efficiency evaluation, real-time CPU and memory power consumption during software execution was measured using tools such as Intel Power Gadget.

TABLE II
TIME TAKEN FOR ASSESSMENTS

Method	Expert	Time Cost (min)	Average Time (min)
Tool measurement	Expert 1	<1	<1
	Expert 2	<1	
	Expert 3	<1	
Manual measurement	Expert 4	25	31
	Expert 5	30	
	Expert 6	38	
Time Saved (min)	-	-	30

In summary, the comparison results indicate (as shown in Table II) that the Software Sustainability Assessment Tool offers significant advantages in terms of efficiency. The tool significantly reduces the time required for the assessment, emphasizing its potential to streamline and enhance the software assessment process

D. Accuracy and Precision Data

TABLE III
ACCURACY DATA FOR SMALL TO MEDIUM-SIZED WEB APPLICATION [31]

Metric	Tool Result	Manual result	Deviation
Maintainability Index (MI)	82.7	81.9	0.8
Package Power (W)	25	24.8	0.2
DRAM Power (W)	3.8	3.7	0.1
CPU Utilization (%)	55	54.5	0.5
Package Temperature (°C)	68	67.5	0.5
Package Frequency (GHz)	2.8	2.75	0.05

TABLE IV
ACCURACY DATA FOR LARGE ENTERPRISE APPLICATION [31]

Metric	Tool Result	Manual result	Deviation
Maintainability Index (MI)	70.5	71.0	0.5
Package Power (W)	40.0	39.8	0.2
DRAM Power (W)	6.5	6.4	0.1
CPU Utilization (%)	75.0	74.5	0.5
Package Temperature (°C)	80.0	79.5	0.5
Package Frequency (GHz)	2.2	2.25	0.05

TABLE V
PRECISION DATA FOR SMALL TO MEDIUM-SIZED WEB APPLICATION

Metric	1	2	3	4	5	Standard Deviation
Maintainability Index(MI)	82.7	82.6	82.8	82.7	82.6	0.07
Package Power(W)	25	25.1	25	25.1	25	0.04
DRAM Power(W)	3.8	3.8	3.7	3.8	3.8	0.05
CPU Utilization(%)	55	55.1	54.9	55	55	0.07
Package Temperature(°C)	68	68.1	67.9	68	68	0.07
Package Frequency(GHz)	2.8	2.8	2.8	2.8	2.8	0.00

TABLE VI
PRECISION DATA FOR LARGE ENTERPRISE APPLICATION

Metric	1	2	3	4	5	Standard Deviation
Maintainability Index (MI)	70.5	70.6	70.4	70.5	70.5	0.07
Package Power(W)	40.0	40.1	39.9	40.0	40.0	0.07
DRAM Power(W)	6.5	6.5	6.4	6.5	6.5	0.05
CPU Utilization (%)	75.0	75.1	74.9	75.0	75.0	0.07
Package Temperature(°C)	80.0	80.1	79.9	80.0	80.0	0.07
Package Frequency (GHz)	2.2	2.2	2.2	2.2	2.2	0.00

The four tables summarize the tool's accuracy and precision. Tables III and IV compare the tool's assessments with manual evaluations, showing minimal deviations, which demonstrate the tool's high accuracy. Table V and VI present the results of multiple runs for small and large applications. Low standard deviations indicate that the tool is highly consistent and reliable across various assessments. These

results validate the tool's effectiveness and stability in evaluating software sustainability.

The evaluation results demonstrated that the tool effectively provided comprehensive and accurate software maintainability and energy efficiency assessments. The generated reports offered valuable insights and actionable recommendations for improving software sustainability, such as optimizing code structure and enhancing energy efficiency. Feedback from users highlighted the tool's ease of use, the clarity of the reports, and the practical applicability of the recommendations.

However, the evaluation also identified areas for improvement. For instance, enhancing the tool's capability to handle larger and more complex codebases efficiently and improving the accuracy of certain metrics could further increase its utility. Continuous feedback and iterative development will be crucial for refining the tool and ensuring it remains up-to-date with the latest advancements in software sustainability assessment.

In summary, the evaluation phase validated the effectiveness of the software sustainability assessment tool in real-world scenarios, providing comprehensive and accurate insights into software maintainability and energy efficiency. The results highlighted the tool's strengths and identified areas for further enhancement, laying the foundation for ongoing improvements and future development.

IV. Conclusion

The conclusion of this project highlights the successful development and validation of a comprehensive software sustainability assessment tool that integrates the Life Cycle Assessment (LCA) with the ISO 25010 quality model. This tool addresses the need for a holistic approach to software sustainability, encompassing environmental, technical, social, and economic dimensions.

The project followed a systematic methodology, beginning with an extensive literature review, integrating theoretical insights into a cohesive framework, and translating this framework into a practical tool through detailed architectural and interface designs. The implementation phase involved setting up the environment, developing the backend and front end, integrating the database, and rigorously testing to ensure reliability and accuracy. Python and PyQt5 were used to create a responsive GUI, and MySQL ensured robust data management.

The evaluation phase demonstrated the tool's effectiveness by applying it to various real-world software projects, including web applications, enterprise systems, and mobile apps. The tool provided detailed maintainability and energy efficiency analyses, offering valuable insights and actionable recommendations for improving software sustainability. User feedback highlighted the tool's ease of use and the practical relevance of the generated reports.

The contributions of this project are significant in the field of software engineering. By developing a model that integrates LCA with ISO 25010, the tool offers a comprehensive assessment of software sustainability that includes environmental impacts. This model bridges the gap between traditional software quality assessments and sustainability metrics, providing a holistic view of software performance. The tool's ability to generate detailed reports with specific recommendations helps developers optimize their code for better maintainability and energy efficiency. Additionally, the tool promotes sustainable software development practices, aligning software engineering with broader environmental and sustainability goals.

Specific contributions include the creation of a maintainability index that incorporates Halstead metrics, cyclomatic complexity, and lines of code, providing a robust measure of software maintainability. Integrating energy efficiency metrics, such as CPU power consumption and DRAM power, offers a comprehensive analysis of the software's environmental impact. The tool's user-friendly interface and robust data management capabilities make it accessible and practical for developers.

Despite its success, the project identified several areas for future improvement. Enhancements to handle more extensive and complex codebases efficiently and improving the accuracy of specific metrics are critical areas for development. Future iterations of the tool should focus on these enhancements to increase its utility and effectiveness. Expanding the tool's capabilities to include more diverse software environments and sustainability goals will further its applicability and relevance. Continuous feedback and iterative development will be essential for refining the tool and ensuring it remains up-to-date with the latest advancements in software sustainability assessment.

Acknowledgment

This research was supported by the Faculty of Computer Science and Information Technology, Universiti Putra Malaysia.

References

[1] V. Grecu, R.-I.-G. Ciobotea, and A. Florea, "Software Application for Organizational Sustainability Performance Assessment," *Sustainability*, vol. 12, no. 11, p. 4435, May 2020, doi:10.3390/su12114435.

[2] C. Calero and M. Piattini, Eds., *Green in Software Engineering*, vol. 3. Cham, Switzerland: Springer, 2015.doi: 10.1007/978-3-319-08581-4.

[3] M. A. H. M. Azman *et al.*, "Life cycle Assessment (LCA) of particleboard: Investigation of the environmental parameters," *Polymers*, vol. 13, no. 13, p. 2043, Jun. 2021, doi:10.3390/polym13132043.

[4] C. Calero, M. F. Bertoa, and M. A. Moraga, "A systematic literature review for software sustainability measures," *2013 2nd International Workshop on Green and Sustainable Software (GREENS), San Francisco, CA, USA,* pp. 46–53, May 2013, doi:10.1109/greens.2013.6606421.

[5] H. Cabezas, C. W. Pawlowski, A. L. Mayer, and N. T. Hoagland, "Sustainability: ecological, social, economic, technological, and systems perspectives," in *Springer eBooks*, 2004, pp. 37–64. doi:10.1007/978-3-662-10270-1_3.

[6] D. Chang, C. K. M. Lee, and C.-H. Chen, "Review of life cycle assessment towards sustainable product development," *Journal of Cleaner Production*, vol. 83, pp. 48–60, Nov. 2014, doi:10.1016/j.jclepro.2014.07.050.

[7] H. Noman, N. A. Mahoto, S. Bhatti, H. A. Abosaq, M. S. A. Reshan, and A. Shaikh, "An exploratory study of software sustainability at early stages of software development," *Sustainability*, vol. 14, no. 14, p. 8596, Jul. 2022, doi: 10.3390/su14148596.

[8] J. Boarim and A. R. C. Da Rocha, "Quality Characteristics of CRM Systems," *SBQS '20: Proceedings of the XIX Brazilian Symposium on Software Quality*, Art. no. 19, Dec. 2020, doi:10.1145/3439961.3439980.

[9] R. Mehra, P. Pathania, V. S. Sharma, V. Kaulgud, S. Podder, and A. P. Burden, "Assessing the Impact of Refactoring Energy-Inefficient Code Patterns on Software Sustainability: An Industry Case Study," *2023 38th IEEE/ACM International Conference on Automated*

Software Engineering (ASE), Luxembourg, Luxembourg, 2023, pp. 1825–1827, Sep. 2023, doi: 10.1109/ase56229.2023.00205.

[10] V. Borja, J. Ávila, M. López-Parra, A. Ramírez-Reivich, and A. Espinosa, "Sustainability Assessment of Products: A Comparative Study of Sustainability Assessment Tools," *Proceedings of the ASME 2014 International Mechanical Engineering Congress and Exposition. Montreal, Quebec, Canada.*, vol. 11, pp. 14–20, Nov. 2014, doi:10.1115/imece2014-39449.

[11] L. M. Hilty and B. Aebischer, Eds., *ICT Innovations for Sustainability*, vol. 310. Cham, Switzerland: Springer International Publishing, 2015. doi: 10.1007/978-3-319-09228-7.

[12] S. Naumann, E. Kern, and M. Dick, "Classifying green Software Engineering - the GREENSOFT model," *Softwaretechnik-Trends*, vol. 33, no. 2, pp. 18–19, May 2013, doi: 10.1007/s40568-013-0027-z.

[13] E. A. Christoforou and P. A. Fokaides, "Life cycle assessment (LCA) of olive husk torrefaction," *Renewable Energy*, vol. 90, pp. 257–266, May 2016, doi: 10.1016/j.renene.2016.01.022.

[14] E. Boakes, J.-K. De Voogd, G. Wauters, and J. Van Caneghem, "The influence of energy output and substitution on the environmental impact of waste-to-energy operation: quantification by means of a case study," *Clean Technologies and Environmental Policy*, vol. 25, no. 1, pp. 253–267, Apr. 2022, doi: 10.1007/s10098-022-02297-y.

[15] C. Celauro, A. Cardella, and M. Guerrieri, "LCA of different construction choices for a Double-Track railway line for sustainability evaluations," *Sustainability*, vol. 15, no. 6, p. 5066, Mar. 2023, doi:10.3390/su15065066.

[16] J. P. Miguel, D. Mauricio, and G. Rodriguez, "A review of software quality models for the evaluation of software products," *International Journal of Software Engineering and Applications*, vol. 5, no. 6, pp. 31–53, Nov. 2014, doi: 10.5121/ijsea.2014.5603.

[17] F. Albertao, J. Xiao, C. Tian, Y. Lu, K. Q. Zhang, and C. Liu, "Measuring the Sustainability Performance of Software Projects," *2010 IEEE 7th International Conference on E-Business Engineering, Shanghai, China*, pp. 369–373, Nov. 2010, doi:10.1109/icebe.2010.26.

[18] B. Penzenstadler *et al.*, "Software Engineering for sustainability: Find the leverage points!," *IEEE Software*, vol. 35, no. 4, pp. 22–33, Jul. 2018, doi: 10.1109/ms.2018.110154908.

[19] C. S. Couto, P. S. Pires, M. S. Valente, R. S. Bigonha, and N. Anquetil, "Predicting software defects with causality tests," *Journal of Systems and Software*, vol. 93, pp. 24–41, Jul. 2014, doi:10.1016/j.jss.2014.01.033.

[20] J. Estdale and E. Georgiadou, "Applying the ISO/IEC 25010 quality models to software product," in *Communications in computer and information science*, 2018, pp. 492–503. doi: 10.1007/978-3-319-97925-0_42.

[21] T. R. D. Saputri and S.-W. Lee, "Integrated framework for incorporating sustainability design in software engineering life-cycle: An empirical study," *Information and Software Technology*, vol. 129, p. 106407, Jan. 2021, doi: 10.1016/j.infsof.2020.106407.

[22] T. Hovorushchenko and O. Pomorova, "Evaluation of mutual influences of software quality characteristics based ISO 25010:2011," *2016 XIth International Scientific and Technical Conference Computer Sciences and Information Technologies (CSIT), Lviv, Ukraine*, pp. 80–83, Sep. 2016, doi: 10.1109/stc-csit.2016.7589874.

[23] N. Condori-Fernandez, P. Lago, M. R. Luaces, and Á. S. Places, "An Action Research for Improving the Sustainability Assessment Framework Instruments," *Sustainability*, vol. 12, no. 4, p. 1682, Feb. 2020, doi: 10.3390/su12041682.

[24] S. Oyedeji, H. Shamshiri, J. Porras, and D. Lammert, "Software sustainability: academic understanding and industry perceptions," in *Lecture notes in business information processing*, 2021, pp. 18–34. doi:10.1007/978-3-030-91983-2_3.

[25] D. Coleman, D. Ash, B. Lowther, and P. Oman, "Using metrics to evaluate software system maintainability," *Computer*, vol. 27, no. 8, pp. 44–49, Aug. 1994, doi: 10.1109/2.303623.

[26] B. Dastjerdi, V. Strezov, M. A. Rajaeifar, R. Kumar, and M. Behnia, "A systematic review on life cycle assessment of different waste to energy valorization technologies," *Journal of Cleaner Production*, vol. 290, p. 125747, Mar. 2021, doi: 10.1016/j.jclepro.2020.125747.

[27] M. Lanza and R. Marinescu, *Object-Oriented Metrics in Practice: Using Software Metrics to Characterize, Evaluate, and Improve the Design of Object-Oriented Systems*. Berlin, Germany: Springer Science & Business Media, 2007. doi: 10.1007/3-540-39538-5.

[28] A. Gurzhii, E. Hanelt, P. Verhoef, and J. Zhu, "Digital Transformation and Flexible Performance Management: A Systematic Literature Review of the Evolution of Performance Measurement Systems," *Global Journal of Flexible Systems Management*, vol. 24, no. 2, pp. 300-317, 2023. doi: 10.1007/s40171-023-00313-4.

[29] J. L. Díaz-Herrera, Ed., *Software Engineering Education: 7th SEI CSEE Conference, San Antonio, Texas, USA, January 5-7, 1994. Proceedings*, vol. 750, Berlin, Germany: Springer-Verlag, 1994. doi:10.1007/BFb0017602.

[30] J. Mancebo, C. Calero, and F. García, "Does maintainability relate to the energy consumption of software? A case study," *Software Quality Journal*, vol. 29, no. 1, pp. 101–127, Jan. 2021, doi: 10.1007/s11219-020-09536-9.