sustainability

MDPI

Article

Practical Sustainable Software Development in Architectural Flexibility for Energy Efficiency Using the Extended Agile Framework

Buerian Soongpol [1], Paniti Netinant [1,*] and Meennapa Rukhiran [2,*]

[1] College of Digital Innovation Technology, Rangsit University, Pathum Thani 12000, Thailand; buerian@gmail.com
[2] Faculty of Social Technology, Rajamangala University of Technology Tawan-ok, Chanthaburi 20110, Thailand
* Correspondence: paniti.n@rsu.ac.th (P.N.); meennapa_ru@rmutto.ac.th (M.R.)

Abstract: Many regular business operations are transforming into digital services, increasing advanced multi-platforms, rapid operational alignment, flexibility, and environmental impact through energy consumption, hardware waste, and technology investments. Flexible and sustainable system development models emphasizing energy efficiency can help innovate software development as digital servicing applications shift. This research is motivated by the need to improve energy consumption in early software design and development due to rising technological efficiency and sustainability demands. Although effective in iterative development and stakeholder engagement, traditional Agile methodologies often struggle with long-term sustainability and energy efficiency. Extended Agile, combining Agile, layered architecture, and aspect-oriented frameworks (ALAI), promises to improve system modularity, flexibility, maintainability, and sustainability. This study's findings are not just theoretical, but also practically relevant, as they explore the energy efficiency of ALAI software development methodologies, using graduate admission information system services (GAISS) as an example. GAISS is a complex system that handles the entire process of graduate admissions, from application submission to final decision. The study quantifies the energy usage of a student-list webpage by analyzing Microsoft IIS server logs from February 2022 to May 2024. Directly applicable findings show that the GAISS based on the ALAI framework reduces energy consumption by 10.7914% compared to traditional Agile software developments. ALAI used 892.80 kWh versus Agile's 1000.80 kWh during operations, saving energy. These findings demonstrate the benefits of integrating aspect-oriented frameworks and layering approaches into Agile methodologies, contributing to sustainable software development discourse. The study emphasizes the importance of energy-efficient frameworks such as ALAI to reduce software systems' environmental impact and promote software development sustainability. The findings of this study, with their practical relevance, assist software developers and organizations in choosing software design and development methods that maximize operational efficiency and environmental sustainability.

Keywords: agile; aspect-oriented; energy efficiency; flexibility; framework; layering; software development; sustainability

Citation: Soongpol, B.; Netinant, P.; Rukhiran, M. Practical Sustainable Software Development in Architectural Flexibility for Energy Efficiency Using the Extended Agile Framework. *Sustainability* **2024**, *16*, 5738. https://doi.org/10.3390/su16135738

Academic Editor: Antonio Caggiano

Received: 17 May 2024
Revised: 3 July 2024
Accepted: 3 July 2024
Published: 4 July 2024

1. Introduction

Software, an integral part of our contemporary society, influences business operations, oversees industrial processes, and facilitates human interactions. The growing ambition of businesses to achieve efficiency, agility, and innovation has sparked a higher demand for advanced and integrated application software [1]. However, this surge in business needs also brings forth significant challenges. Budgets often face strain, and the demands for software development and integration seem endless, with each component requiring meticulous consideration [2]. The potential new business requirements span various areas in software design and development, including compatibility across multiple dimensions [2],

from the complexities of cross-platform compatibility to the subtleties of diverse database architectures [3], the intricacies of front-end and back-end programming languages [4,5], the intricacies of software design and development processes, and sustainability [6,7].

The increasing complexity of the effects caused by the requirements of new businesses on software development and integration has a multifaceted impact on global efforts toward sustainable development. Regrettably, the increase in software development requirements not only causes hardware to consume more energy, contributing to the accumulation of electronic waste (e-waste) [8] but also necessitates proficient developers [9]. The expansion of computing devices, data centers, and servers requires additional energy resources [10], which impedes the achievement of carbon neutrality and contributes to the escalation of greenhouse gas emissions. The proliferation of computers and digital devices has led to a surge in power consumption, specifically the CPU usage [11] and the depletion of batteries [12], adversely affecting users and infrastructure systems. Constant adaptation and iteration of software solutions necessitate significant computing resources, which increases energy usage throughout the software development lifecycle. This trend is extremely concerning and demands immediate action. The sustainability of energy utilization can substantially improve and higher energy efficiency can be guaranteed [13,14] by concentrating on the conceptual evolution and design innovation of new software development solutions that are flexible to expanding business demands and are capable of scaling accordingly. The time has come to make a change.

Previous research has extensively explored energy efficiency solutions for hardware and software designs in various sectors. These include the computational resources employed in software [15,16], software models [14,17], graphical user interfaces (GUI) [16], programming languages [18], encryption [19], cloud computational capabilities [18], and security [11,20]. By addressing these gaps and devising strategies to optimize resource usage and promote the integration of renewable energy sources, developers can pave the way for a more sustainable future. This not only preserves their innovative advantage in the digital domain but also fosters a sense of hope and optimism for a greener world.

This study introduces the Agile-layering aspect-oriented framework (ALAI), a groundbreaking approach designed to enhance sustainability and flexibility in software development. While it shows promising results within our case study, its applicability to other contexts may require further adaptation. This innovative framework aims to revolutionize the integration of Agile methodologies with the robustness of layering and the flexible and adaptable capabilities of the aspect-oriented framework (AOF). The ALAI framework has the potential to significantly enhance the sustainability and architectural flexibility of software systems, thereby supporting the creation of energy-efficient and flexible solutions and environmentally friendly software development. It is particularly suited to the dynamic and diverse needs of modern e-organizations, which require scalable, robust, and sustainable software solutions.

This research seeks to delve into empirical evidence on the effectiveness of the ALAI framework in driving more sustainable and flexible software development practices, with a specific focus on energy efficiency and architectural flexibility. The study is guided by two key research questions:

1. What and how do approaches within the ALAI framework, such as continuous integration and delivery, support Agile software development to succeed in sustainability for energy efficiency? For instance, the ALAI framework encourages the use of automated testing tools to identify and fix energy inefficiencies in the code during the development process.
2. How effectively does this impact sustainable software development for energy efficiency in practical operations?

This study not only contributes to the field but also offers tangible benefits. It fills a crucial gap in existing research by systematically exploring how integrated development frameworks such as ALAI can facilitate the creation of energy-efficient and flexible software architectures. It offers a detailed analysis of the energy efficiency of the ALAI framework

of extended Agile, thus advancing sustainable software development. By demonstrating the practical benefits of integrating sustainability and flexibility in the SDLC, this research inspires academic institutions to transition to more sustainable practices and provides a model that can be adapted for broader industry applications.

The article proceeds as follows: Section 2 presents a comprehensive overview of current software development, spanning Agile methodologies, aspect-oriented frameworks, layering techniques, and energy efficiency considerations in software development. A comprehensive review of the research model and methodology that forms the basis for the conception, construction, and assessment of the pioneering ALAI framework for sustainable software development is provided in Section 3 of this study. The results and discussion are provided in Section 4. The conclusion of the comparison between the Agile and ALAI methodologies for sustainable software development, including the energy efficiency findings obtained from this study, is presented in Section 5.

2. Theoretical Background

The Theoretical Background section provides a comprehensive overview of current software development trends, with a strong focus on Agile methodologies. It also covers aspect-oriented frameworks, layering techniques, and energy efficiency considerations. This section sets the stage by contextualizing the evolution of software development practices and the growing need for sustainable solutions. It delves into the foundational concepts and methodologies that form the basis of the ALAI framework, offering a critical examination of how these approaches contribute to developing flexible, sustainable, and energy-efficient software systems.

2.1. Current Trends in Technological Software Development

The development of software is an industry that is undergoing rapid change due to technological advancements, shifting market dynamics, and increasing consumer expectations [21]. Development teams have increasingly adopted Agile methodologies emphasizing flexibility, continuous improvement, and rapid delivery. This approach has enabled organizations to react more efficiently to shifts in the market and customer input [22]. Agile practices are implemented across multiple facets of software engineering, encompassing pipelines for continuous integration and continuous deployment (CI/CD) [23,24]. These methodologies serve to optimize processes and instill a sense of empowerment in developers, stimulating their imaginations with the potential for accomplishment.

Another significant trend in software development is the shift towards more complex architectures, such as microservices [25] and serverless computing [26]. These architectures, with their superior scalability and flexibility compared to monolithic designs, are not just passing fads. They are tools that empower developers, freeing them to focus on the innovative and business-critical aspects of application development. This shift is creating a sense of excitement about the possibilities of their creations.

Software development is evolving, necessitating a reevaluation of the software construction and delivery processes [25,27]. One example is DevOps, which integrates operations and software development to shorten the development lifecycle while maintaining quality and dependability [24,28]. Another significant development is the integration of artificial intelligence (AI) and machine learning (ML) into software development. This integration is paving the way for more intelligent applications [29] that can automate decisions and predict user behavior, thereby enhancing operational efficiency and user experiences.

However, these developments also bring about complexities, including the increased segmentation and distribution of system architectures (e.g., microservices [25]), the need for ongoing learning and adaptation [2], and the efficiency of team communication and collaboration [30]. Although these obstacles are tangible, they are not beyond hope. Adhering to these principles while upholding quality, user satisfaction, and strategic alignment

is imperative for achieving success in this ever-evolving software capability [31]. This reassurance boosts our capacity to navigate these intricacies effectively.

Table 1 details the progression of software design and development technologies, emphasizing each innovation that has contributed to the present condition of information systems. It is a valuable resource that clarifies the historical backdrop of our discipline by furnishing an exhaustive synopsis of the pivotal technologies and their ramifications on software development.

Table 1. Comparison of technological advancements in software design and development.

Advancement	Description	Impact
Modular programming [32]	An early approach to system development focused on dividing software into separate, interchangeable modules, each handling a specific subtask.	Improved system maintainability and debugging efficiency by isolating functional segments, laying the groundwork for more complex system architectures.
Object-oriented programming (OOP) [33]	Introduced the concept of objects in programming, encapsulating data, and functions into reusable components.	Enhanced code reusability and scalability, allowing for more complex and manageable systems that could be developed with greater efficiency.
Service-oriented architecture (SOA) [34]	An architectural pattern that allows services to communicate over a network through a protocol to perform functions for each other.	Facilitated the development of distributed systems, enabling components to be easily integrated or replaced, thus improving system flexibility and interoperability.
Cloud computing and SaaS [35]	Cloud platforms and Software as a Service (SaaS) models emerged, offering scalable resources and applications over the internet.	Allowed educational systems to scale resources on demand, reduce infrastructure costs, and enhance collaboration and data accessibility from anywhere.
Microservices architecture [36]	An architectural style that structures an application as a collection of loosely coupled services, improving modularity.	Promoted the development of systems that are easier to understand, develop, test, and become more resilient to architecture erosion, supporting continuous integration and deployment practices.
Containerization (e.g., Docker) [37]	Containerization technologies encapsulate software in a complete filesystem that contains everything needed to run code, runtime, system tools, and system libraries.	Enabled consistent development environments, simplified deployment processes, and improved the scalability and portability of applications across different environments.
DevOps practices [38]	Practices that combine software development (Dev) and IT operations (Ops) aimed at shortening the development life cycle and providing continuous delivery.	Fostered a culture of collaboration between development and operations teams, improving the speed, quality, and predictability of development and deployment processes.
CI/CD [23,24]	CI/CD is a method to frequently deliver apps to customers by introducing automation into the stages of application development.	It has enhanced the ability to rapidly iterate and release new features or fixes, ensuring that educational software could evolve quickly in response to user feedback or changing requirements.
Aspect-oriented programming (AOP) [39]	A programming paradigm aims to increase modularity by allowing the separation of cross-cutting concerns.	It enabled more precise and modular code management, particularly for aspects of a system that cut across multiple modules, such as logging and security, thus enhancing system adaptability and maintainability.

The development of software has followed an uncompromising path toward increasingly complex, modular, and scalable architectures that effectively cater to the ever-

changing requirements of contemporary systems, with a specific focus on educational platforms. The progression from modular programming to OOP to more recent methodologies such as microservices and DevOps highlights a more extensive transition towards systems that improve operational effectiveness and promote increased flexibility and adaptability. Nevertheless, to fully exploit the capabilities of these systems, a consistent focus on quality, user contentment, and strategic alignment is required due to their growing intricacy. This strategic alignment not only empowers the audience but also makes them feel responsible for the success of the software systems. The proposed methodology is of utmost importance in this research as it enables the development of resilient, effective, and flexible software systems that flourish in a constantly evolving technological domain.

2.2. Sustainable Computing and Development

Sustainable computing and development represent a critical intersection where technological innovation meets environmental responsibility. Many studies have explored the evolving areas of sustainable computing, emphasizing the need for environmentally conscious practices in software development, hardware design, and digital infrastructure. García-Berná et al. [16] focused on the power growing demand for computational resources in software usage. The performance operations of a health care system using hardware tools were employed and tested by users with many tasks. Data gathered regarding energy efficiency was analyzed, and several characteristics of GUI that contribute to sustainability were identified. Alharbi et al. [40] studied risk management and suggested studying energy-efficient software development in multi-project environments. The experimental methods were used to determine features that are more important in risk assessment. The methods then unanimously agreed on the importance of three features (priority, probability, and risk size). The main features for classifying risk levels are priority, probability, and magnitude of risk, ensured by feature reduction methods. Manimegalai et al. [18] researched the maximization of energy efficiency, comparing different applications and examining current green data center architecture. The study evaluates energy-efficient coding methods in cloud computing and identifies procedures to drastically lower energy usage in cloud environments, highlighting key differences between Java and JavaScript applications and providing optimization of resource utilization and enhancement of energy efficiency. Through code profiling and analysis, specific lines of code, algorithms, and design principles that make applications more energy-efficient are identified, with potential energy savings of up to 17% achievable through energy-efficient coding practices. Alrabaiah et al. [41] introduced Agile Beeswax, a novel agile-based methodology for application development, addressing the challenges and requirements of the mobile development process. Agile Beeswax consists of two main iterative loops, incremental design, and development, structured across six phases. The methodology is developed through a research methodology involving an extensive literature review, interviews with developers, a survey, and proposal formulation. The study revealed that participants predominantly adopted an Agile approach, with 84% utilizing Agile methods in mobile application development. Of these, 56% favored Scrum methodology, approximately 13% each favored XP and Kanban, and 61% considered Agile methods the most effective approach for application development. Akinyele et al. [22] explored simulation and analysis methods for sustainable planning, designing, and developing microgrids utilizing clean energy sources. The simulation framework for sustainable energy planning offers a holistic approach encompassing social, technical, economic, environmental, and policy dimensions. The results demonstrated that existing energy systems software often lacks representation of social parameters, emphasizing the need for integrating social aspects with technical, economic, and environmental parameters in energy systems simulation. By synthesizing previous reviews on sustainable software development, this study identifies research gaps in Agile-based methodologies for accommodating escalating business requirements during incremental software design and development phases.

Table 2 details a variety of software development methodologies that either support sustainability by nature or satisfy functional requirements. Efficiency, waste reduction, and responsiveness are critical factors when developing sustainable software systems. A range of software development methodologies can be designed and employed to advance sustainability in the software development industry.

Table 2. Software development approaches for sustainability software development.

Advancement	Description
Agile development	Agile methodologies emphasize iterative development, flexibility, and stakeholder involvement. This approach can rapidly incorporate sustainability-focused features and practices into the software development process, adapting quickly to changes in environmental standards [42].
Lean software development	Lean principles, with their focus on minimizing waste, directly contribute to sustainability by reducing resource use. This translates into more efficient early design software processes and helps identify unnecessary functionalities that consume extra resources, thereby supporting sustainability [43].
DevOps	Integrating development and operations not only streamlines workflows and reduces redundancies but also significantly increases efficiency. These efficiency gains directly contribute to the sustainability of software development. Automated pipelines in DevOps further promote consistent, efficient, and error-free deployments that reduce waste [44].
Green coding	This approach involves writing code that is as efficient as possible, minimizing the use of computational resources and thus reducing the energy footprint of running software. Green coding practices include optimizing algorithms and reducing complexity [18,33].
AOP	By effectively separating concerns, aspect-oriented programming allows for better modularity and reusability of code, which in turn enhances maintainability and reduces the need for frequent redesigns or resource-intensive updates [2].
SOA	SOA divides functionality into distinct services, which can be managed and scaled independently. This modularity supports energy efficiency by allowing for precise scaling and management of resources according to demand [33].
Open source software	Utilizing open-source software can promote sustainability by encouraging collaborative development and sharing of software solutions, reducing duplicated effort across the industry, and improving the quality and efficiency of software through community contributions [45].
CI/CD	CI/CD practices ensure that updates and improvements are continuously integrated and deployed, which can include sustainability enhancements. This continuous cycle supports incremental improvements in software optimization and responsiveness to sustainability goals [46].

In the ever-evolving field of software development, sustainability is increasingly recognized as a critical objective, driven by the need to reduce environmental impacts. Effective, sustainable software development requires a comprehensive approach encompassing multiple dimensions, including design, process, implementation, deployment, maintenance, and compliance, as stated in [2]. Key elements such as modularity, rapid methodologies, component reusability, and green coding practices play crucial roles in this framework. Furthermore, leveraging advanced technologies such as cloud computing [18] and server virtualization, coupled with strict adherence to environmental regulations, optimizes resource utilization and aligns software development with broader environmental goals. This integrated strategy underscores the commitment to ecological stewardship and positions the software development process sectors as a proactive participant in sustainable innovation, as shown in Table 3.

Table 3. Effective factors for sustainability in software development life cycle.

Dimension	Factor	Description
Design	Modularity	Modular design facilitates easier maintenance, updates, and scalability, reducing overall resource consumption and extending the software's lifespan [32].
	Energy efficiency	Designing software to use fewer computational resources and energy contributes directly to sustainability by minimizing power consumption [47].
Process	Agile methodologies	Agile practices, such as iterative development and continuous feedback, can enhance the adaptability and efficiency of software development, aligning with sustainable practices [48,49].
	Code reusability	Encouraging code reuse speeds up development and decreases the energy and resources needed to create new code from scratch [50].
Implementation	Green coding practices	Coding practices that optimize algorithm performance and resource utilization significantly reduce a software's environmental impact [51].
	Use of sustainable technologies	Incorporating technologies that are inherently more energy-efficient or that support green computing principles contributes to the sustainability of software products [52].
Deployment	Cloud computing	Utilizing cloud services can improve energy efficiency through centralized management and optimized resource allocation, reducing the carbon footprint of running software [53].
	Server virtualization	Virtualization allows for more efficient use of server resources, reducing physical hardware requirements and thus lowering energy consumption [54].
Maintenance	Continuous optimization	Regularly updating and optimizing software to enhance performance and efficiency can prolong the lifespan and sustainability of a software product [55].
	Lifecycle management	Effective software lifecycle management, including planned obsolescence and phase-out strategies, ensures sustainable use and disposal of software products [56].
Policy and compliance	Regulatory compliance	Adhering to environmental and sustainability guidelines and roles can direct software development towards more sustainable practices [57].
	Sustainability metrics and KPIs	Implementing key performance indicators for sustainability helps measure and drive improvements in software development's environmental impact [14].

Software development practices present considerable sustainability obstacles, necessitating substantial waste management and energy consumption. It is imperative to establish a more sustainable future by adopting sustainable practices, which include optimizing algorithms and system and software design and utilizing renewable energy sources to reduce environmental impacts.

2.3. Agile Methodologies and Sustainability

This study employs Agile and Rapid Application Development (RAD) methodologies, both known for their rapidity. Agile focuses on iterative development, flexibility, and continuous feedback, enabling quick adaptation to change. RAD, on the other hand, emphasizes rapid prototyping and feedback, allowing for fast development cycles. These methodologies, with their shared emphasis on rapid delivery, complement each other, providing a balanced framework for efficient and adaptable software development. Agile software development, specifically by utilizing the Rapid Application Development (RAD) framework, presents many advantages for endeavors that necessitate flexibility and expeditious completion [58]. By emphasizing responsiveness and speed, RAD enables software development teams to operate more efficiently and quickly by utilizing iterative releases and ongoing user feedback [59]. Customer satisfaction is enhanced, and the time to market for new software applications is decreased through the active participation of stakeholders and end-users during the entire development process. RAD additionally contributes to mitigating project risks through the early detection and resolution of problems throughout

the development cycle. RAD is also characterized by its emphasis on quality enhancements, achieved through ongoing performance evaluation and user experience [60]. RAD facilitates improved communication and collaboration among stakeholders, developers, and team members, resulting in a more streamlined and effective implementation of projects.

Nevertheless, implementing Agile methodologies for software development maintenance creates obstacles, including scope creep, resource limitations, and cultural assimilation [61]. Agile necessitates frequent iterations and fast responses, factors that may burden resources, result in burnout, and ultimately diminish productivity. Additionally, in environments where standardization and uniformity are critical, the consistency of project outcomes resulting from Agile's flexibility may prove problematic. The Agile software development methodology is characterized by its flexibility, efficiency, and user-centric approach. It guarantees superior software delivery that satisfies and surpasses customers' expectations [62]. The authors provide a concise analysis of Agile software development methodologies and RAD, elucidating the fundamental elements of each approach in Table 4. The comparison concisely outlines the differences between RAD and Agile, emphasizing how each approach handles project management, customer engagement, software development, and team interaction. This comprehension can enable organizations to select the most suitable development methodology under their projects' particular demands and limitations.

Table 4. Comparison of Agile and RAD differentiate in primary features.

Feature	Agile Development	RAD
Focus	Emphasizes iterative development, flexibility, and customer feedback throughout the development cycle.	Focuses on rapid prototyping and quick feedback to develop software quickly.
Development process	Involves sprints or iterations where a small part of the software is built and reviewed continuously.	Uses successive prototyping to refine requirements and solutions quickly, often with little emphasis on planning.
Team collaboration	High degree of collaboration among all stakeholders, including constant communication and regular updates.	Collaboration is centered around feedback on prototypes, with less emphasis on continuous stakeholder interaction throughout each development phase.
Customer involvement	Requires active customer involvement throughout the project for feedback and decision making.	Customer feedback is crucial mainly at the prototyping stage to finalize requirements.
Flexibility	Highly flexible, allowing for changes in project scope and requirements based on feedback and testing.	While flexible, it is often confined to the early stages of development during the prototyping phase.
Documentation	Less emphasis on documentation, more on working software.	Minimal documentation, as the focus is on rapid prototyping and speed.
End product	A potentially shippable product at the end of each iteration, with refinements added in subsequent iterations.	A complete system built in a shorter timeline, with less emphasis on iterative refinement after initial deployment.
Suitability	Suitable for projects where requirements are expected to evolve over time.	Best for projects that require a quick turnaround and where requirements can be modeled through prototypes.
Project size	Scalable and can be adapted for both small and large projects.	Typically, more suited for smaller to medium-sized projects due to the intensive nature of prototyping.
Risk management	Continuous risk management through regular reviews and iterations.	Risk is managed through early user testing and feedback on prototypes.

2.4. Layered Software Architecture

The layering approach is a structural strategy utilized in software development, wherein software is divided into individual, hierarchical layers, and each layer is assigned respect responsibilities [63]. This approach significantly enhances software systems' sustainability, adaptability, and flexibility. By decomposing intricate systems into more manageable components, layering improves the scalability and maintainability of applications [64]. Every individual layer fulfills a distinct purpose [2], including but not limited to data access, business logic, or presentation, and only engages in communication with the adjacent layers. Under this separation of concerns, developers can upgrade or modify a single layer with minimal repercussions on the others; this facilitates maintenance and updates.

An additional critical element of the Layering methodology is adaptability [39]. Software systems must be able to rapidly adapt to changing business environments and emerging requirements without requiring substantial downtime or redevelopment. The layering methodology enables integrating new functionalities or modifying existing ones by concentrating solely on pertinent layers, thereby preserving the system's continuity and efficiency. In general, implementing the layering approach substantially enhances the sustainability of software development.

By dividing software into separate sections assigned particular responsibilities, the layering approach in software development substantially enhances the long-term viability of software systems [65]. This structural organization simplifies maintenance and updates and reduces the risk and complexity associated with bug fixes and updates [2]. In addition to contributing to environmental sustainability, layering enables scalability by permitting distinct layers to expand autonomously in response to fluctuations in demand. This results in increased resource efficiency and decreased environmental impact associated with software system operation, thereby enhancing the environmental responsibility of your work.

The layering approach additionally provides a sustainability benefit through enhanced resource utilization efficiency [66]. Clearly delineated boundaries between layers guarantee that every layer is optimized for its designated purpose, thereby augmenting the system's overall performance and adhering to the principles of green computing. The modular structure of the layering method facilitates the reusability of components, which reduces software development time resources and its carbon footprint. Layering enables the reutilization of pre-existing layers across various applications, thereby diminishing the necessity for reimplementation and, by extension, the resource consumption linked to software development from a sustainability standpoint. Anand et al. [67] suggested that implementing a data access layer across multiple business applications could reduce the need for redundant coding and the energy and materials consumed during testing. Not only are resources conserved by reusing components but the development cycle is also shortened, resulting in a more expedited implementation and diminished ecological footprint.

Additionally, in the contemporary, ever-changing technological environment, the inherent flexibility and adaptability of the layering approach are notably beneficial. In light of evolving business needs and technologies, software systems must be adaptable to seamlessly incorporate new requirements without requiring substantial reengineering [68]. Layering facilitates adaptability by permitting the independent execution of individual layers in response to changes in technologies or business strategies. Furthermore, implementing a modular framework facilitates enhanced integration with developing technologies and external services, granting organizations the flexibility to promptly adapt to changes in software development or technological progressions [69]. The widespread adoption of layering in software development [70], including traditional n-tier applications [71] and more modern microservices architectures [72], demonstrates the innovation and benefits of this approach. Clear classification of responsibilities across layers results in simpler systems to test, debug, and maintain in both instances. In addition, the organization facilitates

scalability by permitting distinct layers to scale independently in response to demand, thus optimizing the utilization of resources and performance throughout the infrastructure.

2.5. Aspect-Oriented Framework (AOF) and Flexibility

AOF is a paradigm that dramatically improves software development's flexibility [39] and sustainability [73]. Separating components from the primary business logic facilitates the development of a codebase that is more structured and feasible to manage. Aspect-oriented approaches (AOAs) reduce maintenance efforts for software projects, such as logging, security, and transaction operations, which frequently affect multiple portions of an application [74], thereby contributing to the sustainability of such projects. By decreasing the time and resources required to maintain existing code and the risk of bugs during updates, this isolation effectively prolongs the lifespan of software systems.

AOF enables organizations to incorporate novel functionality into pre-existing code without requiring modifications [75]. This flexibility is especially beneficial when business requirements are frequently modified, or the application must integrate with evolving external systems; by integrating AOA with other programming paradigms, such as functional or object-oriented programming, the respective strengths of each can be utilized to create more resilient, expandable, and easy-to-maintain [2]. Nonetheless, implementing AOF presents intricacies in comprehending and overseeing the interrelationships among aspects and the core application. This approach may result in an augmented learning curve and potentially complicate the processes of debugging and testing. Notwithstanding these obstacles, the AOF substantially improves software quality by facilitating the adaptability and durability of software development methodologies, promoting more modular, flexible, and maintainable software architectures, and assisting organizations in reacting more promptly to evolving technological advancements and market demands [39,76–79].

2.6. Energy Efficiency in Software Development

Strategic incorporation of energy efficiency principles into the software development process yields numerous advantages. This feature is consistent with the layered architecture of software, which aids in decreasing energy usage and improving the system's sustainability. Energy efficiency is incorporated into architectural decisions starting from the concept design stage, emphasizing modularity and scalability. By making these decisions, one enables future expansions and updates to be implemented with minimal resource consumption and guarantees streamlined functionality in diverse hardware environments housing complex systems. Utilizing data and promoting team collaboration during the early stages of development establishes a fundamental framework that inherently promotes sustainability. As the software advances through the stages of development and implementation, attention is redirected toward combining coding methodologies that maximize energy efficiency. Methodologies containing code refactoring strategies designed to decrease energy consumption are incorporated. During these stages, implementing environmentally friendly technologies, including server virtualization and containerization, enhances energy efficiency by optimizing resource distribution and reducing physical infrastructure requirements.

Strategies such as cloud computing and using energy-efficient data centers are implemented during the deployment phase [18,80]. By employing centralized management and resource sharing, these methodologies effectively mitigate the environmental impact linked to the execution of software applications. Efficient software deployment strategies guarantee the timely delivery of software and its operation within an environment under sustainability objectives. The maintenance phase of software development, which concludes, upholds the dedication to energy efficiency by implementing routine optimizations and updates [52]. Ongoing enhancement of the software guarantees its continued efficacy among evolving user requirements and technological progressions. Furthermore, lifecycle management practices are implemented to guarantee that software can be retired and replaced in an ecologically sound fashion once its operational lifespan concludes.

By systematically implementing energy efficiency measures throughout the software development lifecycle, organizations can guarantee that their products are functional and environmentally sustainable. Table 5 presents an overview of ongoing efforts to enhance the energy efficiency of software development, with a specific focus on each organization's software development processes.

Table 5. Organization concerns energy efficiency in software development layer.

Area	Topics to Discuss
Foundational concepts	Definitions and significance of energy efficiency in software development; relationship between energy efficiency, cost reduction, environmental impact, and system performance [15].
Energy-efficient coding practices	Techniques for optimizing code efficiency, such as using efficient algorithms, minimizing computational complexity, and reducing memory usage; best practices for writing resource-efficient code [50,81].
Design and architecture	The tangible impact of software design and architecture on energy consumption, the benefits of modularity and scalability, and the architectural patterns that actively support energy efficiency (e.g., event-driven architecture, microservices); these are not just theoretical discussions, but real-world factors that can significantly reduce your software's energy footprint [82,83].
Tools and technologies	Effective tools and technologies for monitoring and optimizing energy efficiency, including IDE plugins, profilers, and software analytics tools; these are not just additional complexities, but powerful aids that can assist developers in enhancing energy efficiency [84,85].
Green technologies and practices	Utilization of server virtualization, containerization, and cloud services; a detailed selection process for energy-efficient hardware, considering factors such as power consumption, cooling requirements, and performance; energy-aware deployment practices; examples of green technologies that reduce energy consumption in software operations [86,87].
Agile and CI/CD optimization	Analysis of how Agile methodologies and CI/CD can be adapted for energy efficiency, for instance, by incorporating energy efficiency as a key metric in sprint planning and by continuously monitoring energy usage during the CI/CD process; the impact of iterative development and continuous feedback on reducing waste and resource use in software production and deployment [46,88].
Case studies and practical examples	Insightful real-world examples and case studies that vividly demonstrate the implementation and benefits of energy-efficient practices in software development; these are not just abstract ideas, but practical solutions that have been successfully integrated into companies' development processes [89–91].

3. Research Model and Methodology

This section details the comprehensive research model and methodology used to examine the effectiveness of the ALAI framework in promoting sustainable software development practices. The practical case study centered on the GAISS, a real-world application, serves as a compelling demonstration of the ALAI framework's effectiveness in optimizing system adaptability and reducing energy consumption. The methodology includes a detailed account of data collection techniques, comparative analysis, and the proof of concept (POC) approach used to validate the ALAI framework in a real-world context. This study is guided by two key research questions: how approaches within the ALAI framework, such as continuous integration and delivery, support Agile software development in achieving sustainability and energy efficiency, and how effectively this impacts sustainable software development in practical operations. The hypotheses propose that integrating ALAI framework approaches will significantly enhance Agile software development's sustainability and energy efficiency. The practical implementation of the ALAI framework will demonstrate substantial energy savings and improved adaptability, clearly outperforming traditional Agile methodologies in these aspects.

3.1. Overview of Research Framework and Methodology

The research model and methodology utilized by the authors to examine the effectiveness of the ALAI framework in advancing sustainable software development practices—specifically, energy efficiency and architectural flexibility—are outlined. As a practical case study, our investigation centers on designing and evaluating the GAISS. This methodology developed a sustainable, robust software architecture by extensively integrating extended Agile, AOP, and layering techniques. To optimize system adaptability and reduce energy consumption, the authors of GAISS strategically incorporated sustainability practices at each stage of the software development lifecycle. To evaluate the effects of these interventions, the authors utilized meticulous data collection techniques, including direct measurements of energy consumption and looking into Microsoft IIS server log files; this yielded significant insights into the system's energy efficiency. Furthermore, utilizing a variety of predetermined criteria and analytical methods, the authors conducted a comparative analysis to assess the performance of conventional Agile methodologies in contrast to Sustainable Agile methodologies. The methodology also discusses the obstacles faced and the subsequent adjustments made while implementing environmentally friendly techniques and sustainability principles in software development procedures.

Utilizing this research's POC, methodology significantly substantiated the ALAI framework's efficacy and practical applicability within a realistic environment. The GAISS, initially designed for the university's 2020 graduate open house, was chosen as the case study to ascertain this. Integrating sustainable practices at each development lifecycle phase was the primary objective when designing the GAISS, which aimed to combine conventional and sustainable RAD processes. The approach entailed delineating precise sustainable objectives, such as enhanced resource utilization and decreased energy consumption, and executing these objectives via extended Agile methodologies, augmented with an aspect-oriented framework and layered architecture. This methodology enabled the research to examine numerous hypotheses on architectural flexibility and energy efficiency within a rigorously controlled setting. By comparing real-world performance metrics with anticipated results, the POC approach furnished concrete data regarding the advantages and difficulties of implementing the ALAI framework. This approach validated the research model and presented concrete evidence of the framework's flexibility and scalability in broader software development contexts.

A comprehensive research framework and methodology for sustainable software development is depicted in Figure 1. After identifying the research problem and objectives, this iterative procedure proceeds to construct the ALAI framework, an extension of Agile, along with the integration of layering and AOF methodologies. In the third phase, an educational system case study is chosen to illustrate the practical implementation of the ALAI framework. After completing system design and implementation, data collection and analysis are conducted utilizing log files from the Microsoft IIS server. Conventional Agile methodologies are contrasted with the sustainable Agile approach implemented in GAISS through a comparative analysis. The resolution of challenges and adjustments demonstrates the iterative characteristic of Agile methodologies. The research cycle concludes with the research model's validation and assessment.

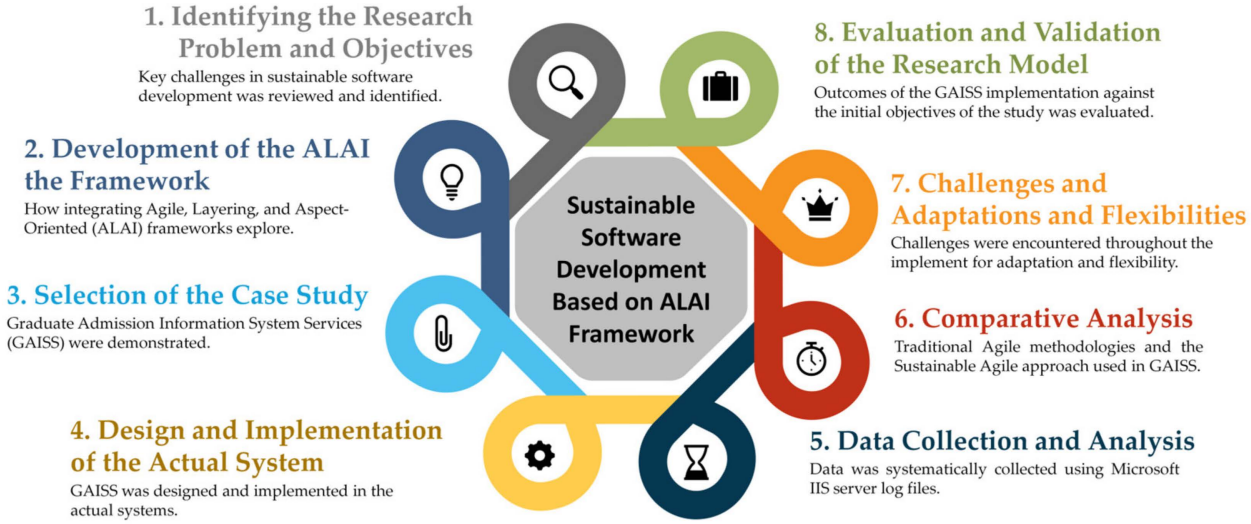

Figure 1. Overview of research methodological steps.

3.2. Graduate Admission Information System Services (GAISS)

GAISS is not just a digital infrastructure for academic establishments, but a significant advancement in managing graduate admissions. Its creation was driven by the need to optimize and enhance the administrative procedures associated with graduate applications. Since its inception, GAISS has proven to be a crucial tool in managing graduate admissions, adapting to the changing technological landscape and the evolving needs of academic institutions.

In 2019, the COVID-19 pandemic profoundly influenced GAISS. The university remained closed in response to the outbreak. Consequently, graduation admission must be accessible around the clock, and services must continue to operate even when the university is closed. Before 2020, the inception of GAISS can be traced back to a goal of substituting labor-intensive, paper-based admission procedures with a digital alternative that is more dependable and efficient. At its inception, the system prioritized fundamental data processing and storage capabilities to efficiently manage graduate applications, student records, and administrative duties. The primary objective of the early revision was to alleviate personnel administrative workload and mitigate the frequent errors that arise from manual data processing. In March 2020, with the increasing integration of digital technologies and the internet into educational administration, GAISS extended its features to encompass a web-based application system known as the preadmission application system. This transition enhanced the ease of access to information for administrators and applicants and implemented primary e-services platforms to aid in advancing graduate programs' educational goals, particularly in recruiting graduate students. As mentioned earlier, the improvements were essential in augmenting the system's versatility, adaptability, and user-friendliness, expanding its functionalities beyond data processing.

The development of GAISS has become increasingly intricate over time, encompassing multiple functions and platforms, necessitating the consideration of numerous data and stakeholder concerns. The augmented demands of the preadmission application system within the student information system presented design layout challenges for several redundant workflows and user interfaces. Furthermore, GAISS has recently implemented additional services, chatbots, and decision workflows, in addition to cloud computing and mobile technology, which are examples of more advanced technologies that have been integrated. Every successive technological development has substantially broadened the functionalities of GAISS. An example of how the implementation of cloud computing enhanced the system is in its increased scalability and accessibility, which enabled administrators and students to access the system from any location and facilitated a seamless integration with other digital platforms. Incorporating supplementary services has en-

hanced the capabilities of GAISS, facilitating more robust data analytics and information system services for faculty and students. These enhancements are important in optimizing the decision-making processes associated with graduate admissions. The development teams have redesigned the GAISS to facilitate complex tasks, primarily by reducing the software design and development phases. Additionally, the ALAI framework addresses sustainability concerns in the GAISS through its layered architecture. The system's layered architecture facilitates adaptability and flexibility in response to evolving technological advancements, student demands, and educational policies. This approach exemplifies the necessity for scalable solutions, as it can efficiently manage substantial quantities of data and user inquiries during periods of high demand. GAISS also necessitates improved sustainability and system maintainability, including simplified updating and maintenance, decreased maintenance expenses, and enhanced software sustainability.

GAISS, which was developed utilizing ALAI software, currently functions as an all-encompassing platform for administration and academic purposes. It supports various functions, such as profiling faculty and students, facilitating course registration and payment, and monitoring performance. The perpetual development of this system serves as evidence of the persistent dedication to adjusting to technological progress and fulfilling the escalating requirements of contemporary educational settings. The GAISS system has emerged as an essential instrument in the management of graduate programs, substantially augmenting the operational efficiency and efficacy of the graduate admission procedure, as well as continuous student oversight. Figure 2 provides a succinct chronology of the software development process for GAISS, spanning various generations, emphasizing significant functionalities, constraints, technological progressions, and the consequences of these developments. GAISS serves as a tangible illustration of ALAI in action, showcasing the advantages of this pioneering methodology for overseeing sustainable, scalable, and complex systems.

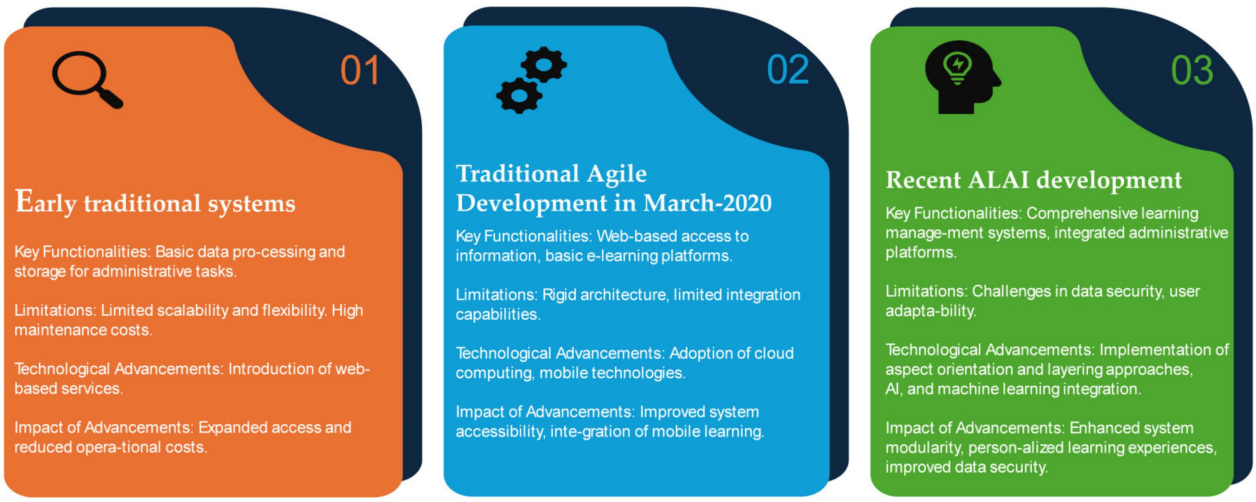

Figure 2. The development time series of GAISS.

3.3. Implementation of GAISS in Traditional Agile Software Development

The implementation of GAISS utilizing conventional Agile software development methodologies is depicted in Figure 3. The iterative process effectively adjusted to the system and its users' changing requirements. The implementation and success of GAISS were significantly influenced by Agile's emphasis on user feedback, particularly in the face of unprecedented obstacles and swift technological advancements such as the COVID-19 pandemic.

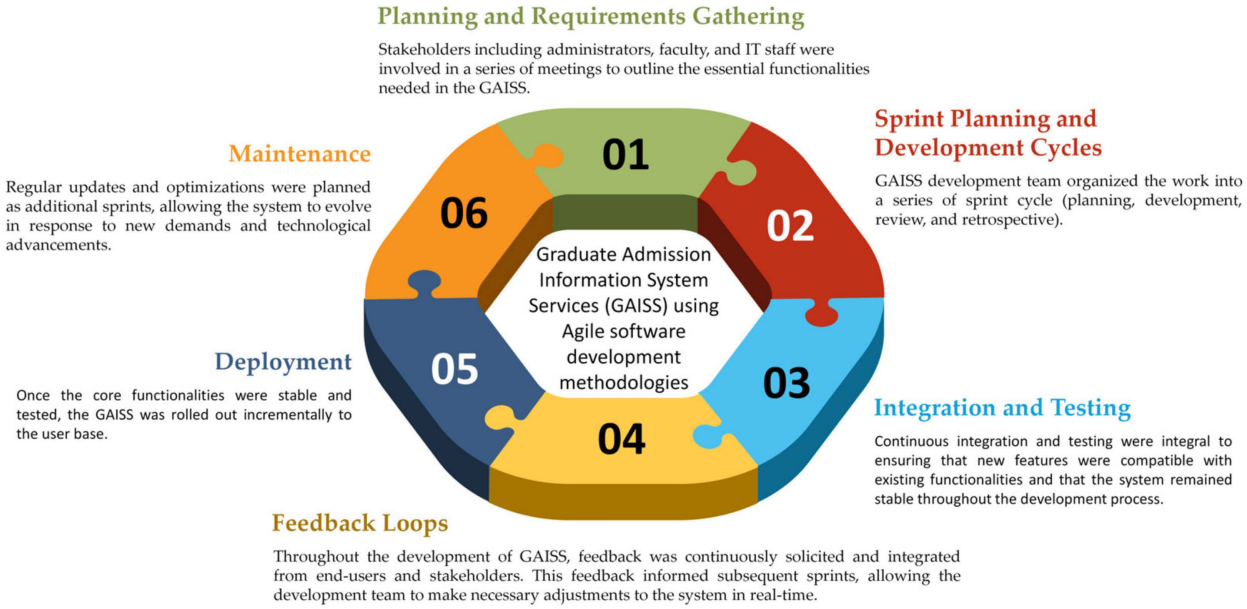

Figure 3. GAISS-utilized traditional Agile software development.

Figure 3 depicts the previous development for GAISS, which employs conventional Agile software development methodologies.

- Initial planning and requirements gathering: The Agile methodology, a collaborative approach, was initiated with a comprehensive planning phase. Key stakeholders, including faculty, administrators, and IT personnel, played a crucial role in these deliberations. Their active participation helped to define the fundamental functionalities essential for the GAISS. These functionalities ranged from basic data storage and processing to more complex requirements such as web access and e-services for student recruitment. During these sessions, the Agile team meticulously documented user stories and created a backlog of prioritized features based on their significance and impact.
- Sprint planning and development cycles: The GAISS development team, guided by Agile principles, established a clear hierarchy of priorities. Each sprint began with a planning meeting, where tasks were assigned from the prioritized backlog. The team operated in two-week cycles, allowing for gradual progress and continuous evaluation of the project's direction. The Agile methodology's adaptability was particularly valuable during this phase, especially when dealing with the sudden shift to remote access requirements due to the pandemic.
- Continuous integration and testing were essential components of the Agile methodology to guarantee the system's stability during development and to ensure that newly added features were compatible with existing functionalities. By employing automated testing scripts, every new module or update was rigorously tested, enabling the team to address any issues that arose promptly. By conducting continuous testing, GAISS was able to preserve a superior standard of quality and functionality, thereby circumventing the drawbacks associated with conventional waterfall software development methodologies that postpone testing until project completion.
- Iteration and feedback loops: The Agile framework is renowned for its iterative process and its reliance on user feedback. Throughout the development of GAISS, end-user, and stakeholder feedback was consistently collected and incorporated. This feedback played a pivotal role in shaping subsequent sprints, allowing the development team to make real-time system adjustments. For instance, in response to initial users' feedback on the usability of the web-based access, the development team could promptly prioritize these modifications in the next sprint.

- Software deployment commenced once the GAISS had been gradually implemented among the user base after the validation and optimization of its fundamental functionalities. Agile's emphasis on continuous delivery made the controlled deployment of the system possible; necessary adjustments were implemented in response to user interactions and system performance.
- Continual maintenance: The Agile methodology continued to influence GAISS's maintenance phase after deployment. System evolution was facilitated by incorporating periodic updates and optimizations during additional sprints, which responded to emerging demands and technological developments. Agile frameworks enabled the project to maintain adaptability, user-centricity, and resilience in swiftly evolving external circumstances.

3.4. Data Collection

This research aims to compare the energy efficiency of two GAISS models, one of which was constructed utilizing the conventional Agile model and the other through the ALAI framework. Both models are deployed on Microsoft IIS web servers, which are robust and widely utilized in enterprise environments, as part of the data collection procedure. A significant volume of data was collected for 825 days, starting on 10 February 2022 and ending on 15 May 2024. This data, collected from real-world scenarios, enabled a comprehensive analysis of the systems through a unified student-list webpage, adding practicality and relevance to the research. An ordinary undertaking demonstrated the efficacy of each approach. The system accessing log files comprises non-relational store data, including the following: date, time, operating system, location, web type, IP address, speech of access, and time of access. The data for this investigation, centered on energy measurements, were computed using the time access per second on web access for student lists. Integrated tools with the MS IIS web server were utilized to monitor energy consumption during the development and operation of the software. Preparations are underway to analyze the data collected during this period to compare the energy efficiency of the two software development models. The rigorous data collection methodology employed in this study guarantees an accurate evaluation of the impact of the ALAI framework compared to traditional Agile approaches on energy consumption. This data establishes a solid basis for identifying which methodologies might significantly enhance operational efficiency and sustainability in software development.

3.5. Comparative Analysis Methodology

Given their pivotal role in software engineering, the ALAI and GAISS models demand a thorough investigation to truly grasp their influence on software operations' flexibility and energy efficiency. This evaluation will delve into energy usage, architectural flexibility, and system performance. Energy efficiency criteria encompass a range of factors, including increased energy usage, energy per transaction or user session, and total energy consumption throughout the development and operation phases. The level of architectural flexibility is gauged by the ease of implementing changes and the impact on system services. A system's energy consumption over time and its ability to adapt its software to new requirements with minimal revision are key indicators of sustainability. To gauge the significance of the observed differences between the two development methodologies, statistical analysis is imperative. Descriptive and inferential statistics are employed to determine the significance of the disparities and provide a succinct data summary.

The authors adopt a unique, systematic approach that blends theoretical analysis and empirical investigation to authenticate the effectiveness and efficiency of the ALAI methodology. To establish ALAI's credibility, a comprehensive research strategy is deployed, spanning multiple phases, which includes the collection and analysis of both qualitative and quantitative data. This study aims to evaluate ALAI's impact on software development, with a focus on key performance indicators, such as energy conservation and integration of servicing development into the system.

The comparative analysis offers a comprehensive synthesis of the qualitative assessment findings, providing a complete view of the ALAI framework's advantages over traditional Agile methodologies. This analysis involves integrating the results of statistical tests with insights gleaned from qualitative data, such as users' and developers' perspectives on system flexibility and maintenance needs. The aim of this comprehensive analysis is to identify patterns and trends that shed light on how one methodology can outperform another in specific areas, such as system flexibility or energy efficiency.

As demonstrated by Equations (1) and (2), the ALAI framework's influence on energy consumption during software design and development is quantified using a combination of metrics and factors. This approach provides a clear and quantifiable measure of the ALAI framework's impact, enhancing the readers' understanding of its relevance to the research.

$$\text{ALAI Energy Impact} = \text{Energy Traditional} - \text{Energy ALAI} \quad (1)$$

where: Energy Traditional is the energy consumption of a similar software system developed using traditional methodologies without the ALAI framework.

Energy ALAI is the energy consumption of a software system developed using the ALAI framework.

$$\text{Energy per accessibility} = \text{Average accessibility processing time} \times \text{Average Energy use per access} \quad (2)$$

where: Average accessibility processing time is the total average time to process a single accessibility request on the web application. This time is typically measured in seconds or milliseconds and represents the duration from when a request is initiated until the application completes the request.

Average energy use per access is the average amount of energy consumed by the web application to handle a single access or request. The unit is usually measured in watt-seconds (watts) or joules, as both these units denote energy used over a period.

Equation (1) calculates energy consumption differences between ALAI-based and non-ALAI systems. Positive values indicate lower energy consumption from the ALAI framework, while negative values indicate higher energy consumption. To ensure accurate measurements, this study uses a range of metrics including power usage (watts), CPU utilization (percentage), memory usage (megabytes), network traffic (kilobytes per second), and system efficiency (transactions per watt). These metrics, collected using software and hardware monitoring tools, sensors, and energy meters, form the basis of our findings.

4. Results and Discussion

This section presents the results and discussion, focusing on implementing sustainable practices in software development and architectural design for sustainability and flexibility. This section analyzes the empirical data collected from GAISS, highlighting the energy efficiency and performance improvements achieved through the ALAI framework. The discussion not only delves into the practical benefits of integrating sustainability into software development but also inspires and motivates by comparing the traditional Agile approach with the ALAI methodology and offering insights into the challenges and adaptations necessary for implementing such frameworks.

4.1. Implementation of Sustainable Practices in Software Development

Significant changes have occurred in recent decades concerning the evolution of educational technology, specifically in the development of administrative and academic systems. These changes have been primarily caused by the ongoing progress of technology and the growing sophistication of educational operations. At the outset, educational systems employed conventional Agile methodologies, which, although groundbreaking, frequently required assistance with scalability, flexibility, and adaptability on account of the time-consuming nature of manual processes. The constraints were substantial as

academic establishments demanded increasingly sophisticated functionalities to manage the expanding variety of service modalities and complex operational requirements. With its emphasis on scalability, adaptability, and flexibility, the ALAI methodology presents a potentially viable resolution to these obstacles.

As a reaction to these obstacles, educational system software development initiated the integration of more sophisticated methodologies, including aspect orientation and layering approaches. The ALAI methodology, which serves as a promising framework for this investigation, symbolizes a paradigm shift, as depicted in Figure 4. The framework expands upon conventional Agile software development methodologies by integrating sustainability considerations, which are critical for contemporary systems that prioritize rapid development, high modularity, and energy efficiency. The ALAI methodology merges Agile methodologies, the AOF, and layering techniques focusing on rapid development and reusability. This theoretical framework is not just a promising solution, but one that is highly compatible with stakeholders' requirements in educational environments, providing a sense of reassurance about its suitability.

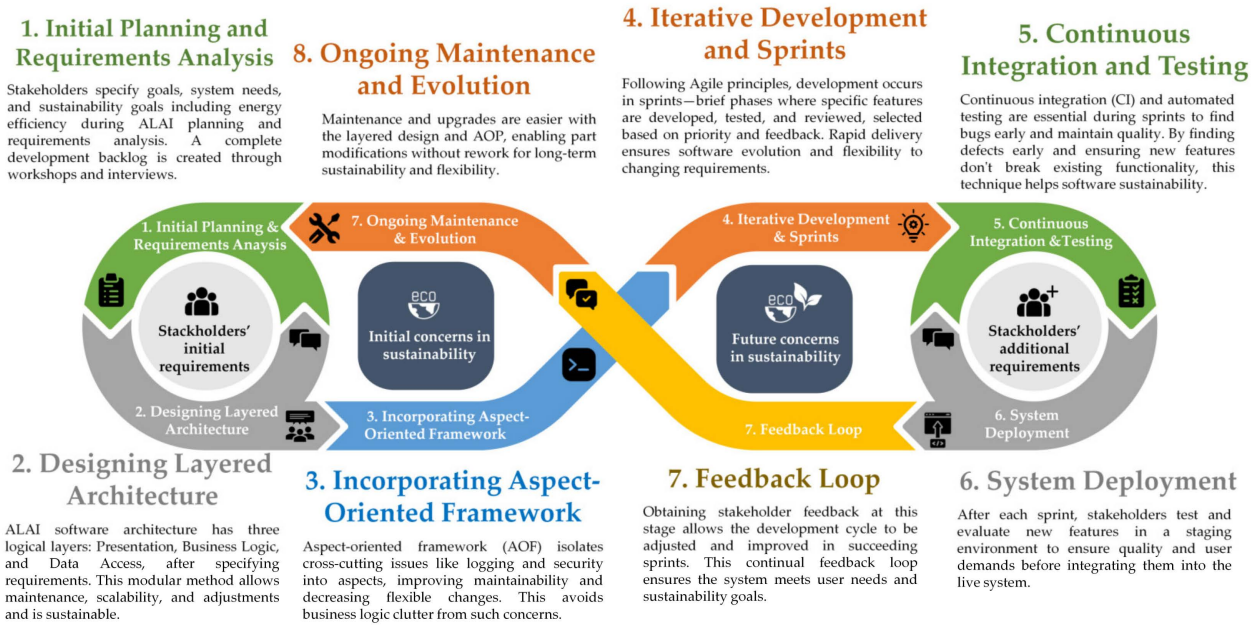

Figure 4. ALAI software development methodologies, an extended Agile software development.

The AOF further improves the capability of software development by enhancing the separation of concerns. Consequently, distinct facets of the software are managed in isolation, including data management, user interface development, and security, particularly on ISO27001:2013, Information Security Management System. This architecture facilitates comprehension and modification of the system. As supported in [2], the framework enables increased modularity by effectively isolating cross-cutting concerns from the core business logic, facilitating more flexible and manageable system modifications in response to emerging requirements. In contrast, the layering methodology applies well-defined levels of accountability to architectural applications, resulting in enhanced organizational understanding, scalability, and maintainability of the systems. As an illustration, alterations to the user interface layer do not impact the data management layer, thereby enhancing the adaptability and maintenance simplicity of the system. The structural method contributes to the development process's effectiveness and efficiency and guarantees system services' sustainability and continuity.

By implementing the preadmission application system with the ALAI methodology, the university can construct information systems that are more robust, flexible, and adaptable. However, it is essential to note that ALAI methodology implementation may

necessitate substantial modifications to the current software development processes and may subject the development team to a learning curve. Developed with your requirements, these systems are adaptable to current demands and resilient enough to adapt to future technological advancements and alterations in organizational requirements. By adopting this all-encompassing strategy, academic establishments can efficiently utilize their digital infrastructure to improve administrative and academic operations while adhering to sustainable software development tenets.

The ALAI methodology, a unique blend of Agile development, layered software architecture, and AOFs, is meticulously designed to create flexible and eco-friendly complex software systems. This innovative approach, depicted in Figure 4, kicks off with a crucial initial stage of planning and requirements analysis. Here, stakeholders collaborate to define objectives, system requirements, and sustainability goals, such as durability and energy efficiency. This stage lays the groundwork for a comprehensive requirement backlog that ensures a successful software development project by prioritizing the establishment of specific, actionable requirements.

After the requirement gathering, the ALAI methodology moves into the layered architecture phase. This stage organizes the system into logical layers, each with a specific purpose, such as data access, business logic, or presentation. This approach not only enhances modularity and simplifies system maintenance but also promotes scalability and future modifications, significantly enhancing the system's sustainability.

In the Incorporating AOF phase, which follows architectural design, cross-cutting concerns, including transaction management, logging, and security, are isolated from the business logic. Separating these concerns into separate modules or aspects prevents them from interfering with the primary functionalities. As a result, maintainability is enhanced, and the likelihood of encountering bugs during iterative development phases is diminished.

Subsequently, the ALAI methodology advances towards sprints and iterative development following Agile principles. In this methodology, the development process is divided into sprints during which specific features are created, tested, and evaluated under stakeholder and priority feedback from previous sprints. This component guarantees that the software undergoes ongoing development and swiftly adjusts to emerging requirements to fulfill their ever-changing requirements.

The experiments employ rigorous testing methods, whether manual or automated, throughout the continuous integration and testing development phases to ensure early detection of integration issues and consistent software quality. Continuous testing is essential in maintaining software quality and detecting defects early, thereby instilling confidence in the software's integrity.

The ALAI methodology enters a crucial phase after development, where stakeholder involvement is key. The software is deployed in a staging environment during the system deployment phase to facilitate stakeholder evaluation. This stage serves as a vital platform for gathering stakeholder feedback, which is then incorporated into subsequent development sprints. By ensuring that the software consistently meets user requirements and aligns with long-term sustainability goals, this iterative feedback loop underscores the importance of stakeholders in the software's success.

Ultimately, the software's functionality and current status are preserved through the continuous maintenance and evolution stage. The streamlined structure and distinct compartmentalization of components enable more convenient updates and alterations that do not require substantial redeployment, enhancing the system's enduring durability and flexibility.

This methodology satisfies the constantly shifting demands of contemporary software development. It guarantees that the developed systems are resilient, flexible, and environmentally friendly, per the objectives delineated in the ALAI methodology.

4.2. Architectural Software Design for Sustainability and Flexibility

The GAISS architecture, meticulously designed to cover all aspects of the system, instills confidence in its efficiency and sustainability. Each layer fulfills a distinct purpose, ensuring the system operates in an environmentally friendly and streamlined manner from its very inception. The user interface, which interacts with end-users, and the hardware that powers the system are both part of this comprehensive design. The sustainability dimension of the GAISS architecture is underscored in Figure 5, emphasizing its inception to promote efficiency, flexibility, and sustainability.

Figure 5. Architectural software design for sustainability and flexibility.

The hardware layer of GAISS, a fortress composed of necessary physical components, plays a crucial role in promoting sustainability. The graphics processing unit (GPU) handles all graphics-related operations, while the central processing unit (CPU) acts as the system's brain. Networking components, such as switches and routers, ensure the smooth transmission of data. Most importantly, the power supply unit provides the electrical energy required by all hardware components. GAISS's commitment to diminishing operational impact and promoting sustainability is evident in its emphasis on energy-efficient models.

In the GAISS system, the system design utilizes cloud technologies at the hardware abstraction layer (HAL) to facilitate the elastic scaling of resources such as storage and processing power. The system's elasticity enables it to adjust to fluctuating loads without requiring modifications to its physical hardware. Hardware monitoring tools are integral to this layer, furnishing instantaneous insights into tangible devices' operational status and health. These tools are crucial in preserving system dependability and optimizing energy consumption. An additional pivotal element is the implementation of virtual machines (VMs). VMs provide a flexible environment by enabling the concurrent operation of multiple instances of operating systems on a single physical machine. This feature improves the flexibility and utilization efficiency of the hardware resources, thereby enhancing the overall efficiency of the GAISS architecture.

In the GAISS architecture, the software system layer comprises the operating systems responsible for hardware resource management and service provision to the executing software. As an illustration, database management systems (DBMS) are essential in storing, retrieving, administering, and manipulating data. They ensure the security and integrity of data. In contrast, network operating systems (NOS) are focused exclusively on managing network connections. Drivers enable communication between hardware devices and the operating system. Utility software offers a system's supplementary functionalities,

including system monitoring, data backup, and security services. These are indispensable for the GAISS architecture to preserve system integrity and operational efficiency.

The middleware/service layer in the GAISS architecture comprises several critical components. Backup systems, for instance, are crucial in guaranteeing data integrity and facilitating recovery operations. MS IIS servers, on the other hand, are responsible for hosting web applications and processing requests, contributing significantly to the scalability and security of web services. Containers, as an alternative to VMs, encapsulate an application and its dependencies within a portable runtime environment, facilitating operations related to scaling and deployment. Service-oriented architecture (SOA) components in this layer facilitate the decomposition of applications into more minor, seamlessly integrable services, enhancing flexibility and reusability within the GAISS framework.

The sustainability layer in the GAISS architecture is a testament to our dedication to promoting sustainable energy efficiency. Situated in a critical intermediary position between the functional and aspectual layers and the middleware/service layer, this stratum integrates sustainability principles into the operational middleware services and the fundamental business logic of the application. By consistently enforcing energy-efficient practices and sustainability metrics throughout the upper layers of the architecture, the sustainability layer guarantees the integration of sustainability principles into the system's day-to-day operations. It does more than just energy consumption optimization; it utilizes advanced algorithms and predictive analytics to dynamically modify resource allocation and anticipate forthcoming loads, thereby improving the efficiency and preparedness of the system. It works with the hardware abstraction layer to guarantee the most efficient utilization of cloud and virtual resources, thereby minimizing wastage and encouraging the adoption of renewable energy sources whenever feasible. Furthermore, it oversees the system's compliance with regulatory standards and environmental objectives, accounting for sustainability reporting and facilitating green IT practices such as electronic waste recycling. By employing these all-encompassing tactics, the sustainability layer motivates us to achieve our CSR objectives and positions us to control operational expenses while complying with rigorous environmental policies more effectively; thus, it strengthens our dedication to sustainability throughout our business.

The functional and aspectual layers in the GAISS architecture fulfill separate objectives. For example, the functional components manage the fundamental business logic unique to GAISS. This layer includes application management, payment processing, and user data management. Conversely, the aspectual components, which employ aspect-oriented programming methodologies, tackle issues that transcend various application domains. These concerns include logging, error handling, security, and transaction management. The aspectual components enhance the enterprise logic's maintainability and overall efficiency of the GAISS architecture by consistently attending to these concerns without introducing chaos.

The integration business logic layer in the GAISS architecture is where the functional and aspectual components interlace, creating a unified and effective system. This integration promotes seamless functioning and upkeep by ensuring that every facet of the application logic is coordinated and efficiently controlled.

The data access layer in the GAISS architecture is critical in connecting the business applications to the physical databases. Its principal obligation is to guarantee the effective retrieval and storage of data. The data access layer in the GAISS architecture is critical in connecting the business applications to the physical databases. Its principal obligation is to guarantee the effective retrieval and storage of data. The data access layer accomplishes this by optimizing user experience and interaction across multiple platforms and devices, which is crucial. This layer mitigates energy usage associated with client-side processing by implementing streamlined coding techniques to reduce data transfer and load times. Prioritizing user-friendliness enhances the overall user experience and demonstrates efficiency in design.

In conclusion, the interface layer serves as the gateway through which users interact with GAISS. Under its intuitive and responsive design, this layer embodies our unwavering commitment to user-centric design. By optimizing user experience and interaction across multiple platforms and devices, we ensure our users feel valued and attended to. Additionally, this layer plays a crucial role in mitigating the energy usage associated with client-side processing by implementing streamlined coding techniques to reduce data transfer and load durations. By prioritizing user-friendliness, we not only enhance the overall user experience but also demonstrate respect for the time and effort of our users.

4.3. Challenges of Flexibility and Adaptations

The ALAI methodology is a practical approach that enables software developers to augment the flexibility and adaptability of their systems, transcending its status as a mere theoretical concept. Through its focus on these fundamental prerequisites, ALAI theory offers a pragmatic structure for developing flexible software for evolving technologies and needs. In this analysis, the result shall explore the specific ways in which ALAI tackles the concepts of flexibility and adaptability:

One of the key benefits of ALAI is its intuitive methodology for separating concerns. The ALAI AOF empowers developers to modify or enhance specific system components, such as those for data access, searching, updating, deleting, sorting, logging, and error handling, without impacting the core application logic. This decoupling of components not only simplifies code management but also increases the system's flexibility. Developers can make changes to a single component without worrying about the impact on others, thereby reducing the fear of causing system-wide disruptions.

Flexibility is not just a possibility, but a reality with ALAI's layered architecture approach. This approach partitions the application into individual layers (e.g., presentation, business logic, data access), each provided with a unique set of responsibilities. The separation enables modifications or updates to be implemented in a single layer without impacting the operation of the remaining layers. As an illustration, modifications to the presentation layer can be implemented without causing any disruption to the business logic or data access layers, thereby not just enhancing, but revolutionizing the efficiency and flexibility of the development process. Flexibility is a key aspect of ALAI, and it is achieved through the utilization of Agile methodologies. These methodologies, including iterative development, ongoing feedback, and task prioritization, are at the core of ALAI. They enable swift modifications to a project's scope, features, and priorities. In a recent project, the study was able to substantially modify stakeholder demands while ensuring uninterrupted development. The ability to adapt to evolving stakeholder demands and market circumstances is crucial, as it allows the development process to adjust as necessary without encountering significant obstacles.

Adaptable development cycles are capable of fostering adaptability. The ALAI components guarantee a responsive and adaptable development process by implementing periodic reviews and adjustments after each iteration. Adaptability is paramount in evolving the product under emerging trends and user feedback. A modular design is facilitated by the AOF and the Layering principle, which entails encapsulating components or functionalities within discrete modules or aspects. As an illustration, the AOF facilitates the encapsulation of cross-cutting concerns. In contrast, the Layering principle assigns distinct responsibilities to each layer, enhancing the system's flexibility in response to evolving technologies or requirements.

Figure 6 shows that Agile and ALAI methodologies in the services portal substantially improve flexibility and adaptability.

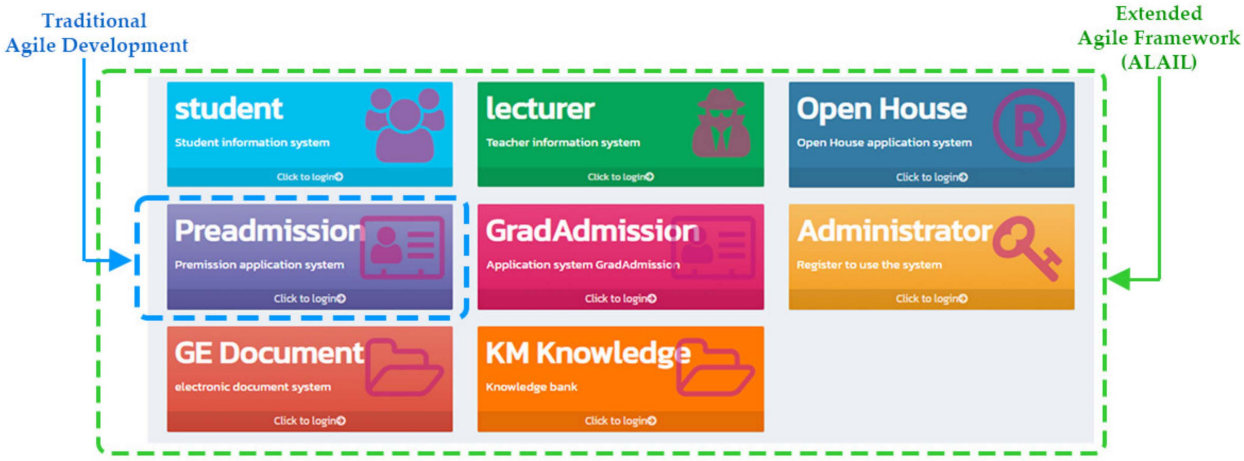

Figure 6. Portal figure of GAISS.

This is exemplified by the Agile-driven Preadmission system, which promotes rapid change adaptability—a critical attribute for systems that necessitate ongoing updates in response to evolving university policies or user feedback. By employing traditional Agile methodology, the Preadmission system can be rapidly adapted to accommodate evolving requirements without requiring substantial redevelopment. On the contrary, alternative services such as GradAdmission and GE Document derive advantages from the ALAI methodology, a robust implement that integrates the extended Agile with improved architectural flexibility via AOF and layering. By enabling system developers to independently modify or expand individual layers, including business logic, presentation, and data access, this approach enhances the flexibility and scalability of the system, as illustrated in Figure 7.

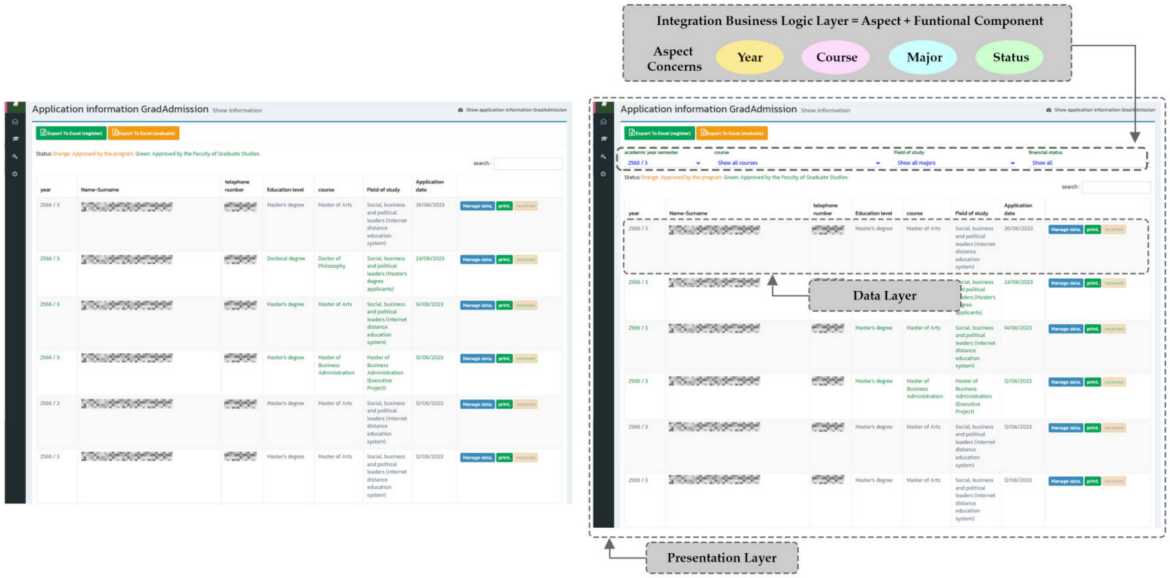

Figure 7. Student-list webpage using Agile, on the left, and ALAI, on the right.

AOF, a cornerstone of the ALAI methodology, is essential in preserving an integrated and unified user experience throughout the portal. This architecture enables the creation and ongoing management of cross-cutting features such as logging and security to be conducted independently, thereby preventing any disruption to the fundamental operations of the application. This architectural framework enables the iterative development of every service, encompassing Open House applications and Student Information Systems, following technological advancements and user feedback. Consequently, it establishes a

solid basis for ongoing enhancements. This approach provides customized capabilities to meet particular requirements, such as disseminating knowledge or managing complex documents. Additionally, the ALAI methodology substantially improves system availability, user satisfaction, and service delivery.

The GAISS architecture exhibits the capabilities of the ALAI and Agile methodologies. The fundamental components of the system, namely the hardware and hardware abstraction layers, guarantee scalability and operational autonomy from particular hardware configurations. This attribute is significant for accommodating expansion and technological advancements without requiring comprehensive system redesigns. The software system and middleware layers are critical components in facilitating the distributed services of the system by ensuring secure and efficient data communication and resource management. Through modular design and separation of concerns, the application framework and business logic layers enable rapid development and effortless adjustment to the evolving requirements. AOF is primarily used to handle cross-cutting concerns such as logging and security independently of the leading business logic.

The GAISS architecture is designed to optimize system performance in high-demand environments. The data access layer, a critical component, is responsible for optimizing database interactions, thereby enhancing system efficiency. The user interface, or UI, layer, on the other hand, prioritizes providing an intuitive and responsive user experience customized to suit the requirements of various user demographics. By meticulously structuring the layers, the current functionalities are supported, and the system is guaranteed to be prepared for future technological advancements and alterations in user requirements, such as dynamic multi-language support [92]. By incorporating Agile and ALAI methodologies into various services such as Preadmission, GradAdmission, and KM Knowledge, the system can be dynamically flexible to evolving educational policies and user input. This strategic approach ensures the scalability and resilience of the system. The flexibility and adaptability of the portal are ensured, allowing each service to effectively fulfill particular requirements while preserving a unified and consistent user experience. ALAI arms the system against forthcoming technological developments and shifting user demands by enabling a clear separation of concerns and a modular architecture. The architectural design considers future needs and facilitates effortless adaptation, ensuring the software's durability and continued applicability.

4.4. ALAI vs. Agile Software Development

In this section, the quantitative results are presented. Within the rapidly evolving domain of software development, selecting a methodology can profoundly affect the longevity, effectiveness, and flexibility of software projects. Both the ALAI methodology and conventional Agile approaches present unique benefits and obstacles. Therefore, developers must thoroughly understand these distinctions before determining their following actions. In order to facilitate comprehension, Table 6 presents an exhaustive comparison between ALAI and conventional Agile spanning multiple facets, including but not limited to the following: concept, procedural aspects, sustainability considerations, adaptability, modularity, software architecture, constraints, benefits, and appropriateness for projects of varying scales. This brief and comprehensible comparison aims to underscore how each methodology tackles the complex issues of contemporary software development and corresponds with particular project specifications. This aids critical players in determining which approach is most suitable to address their specific development requirements.

Table 6. Comparison between ALAI vs Agile software development.

Criteria	ALAI Methodology	Traditional Agile
Concept	Integrates Agile practices with layered architecture and aspect-oriented programming to enhance sustainability, flexibility, and modularity in software development.	Focuses on iterative development and continuous feedback to quickly adapt to changes and deliver functional software rapidly.
Steps	Includes stages such as initial planning, designing layered architecture, incorporating aspect-oriented programming, iterative development sprints, continuous integration and testing, and feedback loops.	Involves continuous iterations of planning, coding, testing, and reviewing within set time frames (sprints), focusing heavily on user feedback and rapid delivery of features.
Sustainability Concerns	Explicitly addresses sustainability by optimizing for energy efficiency and resource management through design and architecture choices, such as separating cross-cutting concerns and creating efficient layers.	Sustainability is not a primary focus, though Agile practices of minimizing waste and focusing on necessary features indirectly support sustainable development.
Flexibility	High flexibility achieved by separating concerns using aspect-oriented programming and by using a layered architecture, which allows easy updates and scalability without extensive system-wide modifications.	Agile is inherently flexible due to its iterative nature and responsiveness to change but lacks specific structural mechanisms for handling large-scale modifications easily.
Modularity	Promotes modularity through the use of layered architecture and aspect-oriented programming, which isolates different functionalities and cross-cutting concerns into separate, manageable sections.	Modularity depends on the team's approach to software design and is not specifically dictated by Agile methodologies; often, modularity must be consciously implemented.
Software Architecture	ALAI mandates a specific software architecture approach by incorporating layers and aspects to systematically manage complexity and improve maintainability.	Traditional Agile does not prescribe a specific architecture; it allows teams the flexibility to choose the architecture that best suits their project needs.
Limitations	ALAI can be complex to implement due to the need for specific expertise in aspect-oriented programming and understanding of layered architecture. It may also require more initial planning and setup time.	Traditional Agile might struggle with large, complex systems where changes have broad impacts, and can suffer from lack of long-term planning and potential for accumulating debt.
Advantages	ALAI provides a structured framework that can handle complex, scalable projects more effectively, with clear separation of concerns that enhances maintainability and allows targeted optimizations.	Agile offers high adaptability, rapid development cycles, and strong stakeholder engagement, which can lead to high customer satisfaction and immediate problem resolution.
Project size	Scalable and can be adapted for both small and large projects.	Typically, more suited to smaller- to medium-sized projects due to the intensive nature of prototyping.

4.5. Descriptive Analysis

A comprehensive evaluation of the energy consumption of the student-list webpage is necessary for comprehending the effectiveness and efficiency of this critical element within the GAISS. This segment provides an extensive analysis of the log file information gathered from the MS IIS server, specifically emphasizing crucial metrics, including the frequency of accesses, duration of responses, volumes of data transfers, and distribution of users, as shown in Table 7. The authors aim to identify patterns and trends that impact energy consumption by analyzing these qualities. Furthermore, this study investigates the consequences of many variables, such as session durations, HTTP status codes, and user agents, on the overall energy efficiency of the student-list webpage. The present analysis offers significant insights into the operational dynamics of the webpage, shedding light on

potential optimization areas that could not only improve energy efficiency but also enhance system performance, leading to a more sustainable and effective operation.

Table 7. Descriptive data for quantitative analysis.

Metric	Description	Value
Prospective students	2022	355
	2023	1073
	2024	250
Registered Students	2022	319
	2023	1054
	2024	247
System Users	Super Users	3
	Department Administrators	74
	Users	9
Access Metrics	Total Number of Accesses	946,103
	Frequency of Access (Daily Average)	1213
Access Methods	POST	460,929
	GET	424,901
	HEAD	57,344
	OPTIONS	2815
	PUT	58
Performance Metrics	2022 Average Response Time (ms)	699.2
	2023 Average Response Time (ms)	732.5
	2024 Average Response Time (ms)	416.2
URL Path (Top 5)	/GradAdmission/RegisterList.aspx	121,080
	/LabelGen.ashx	110,778
	/admin/ManageScholarship.aspx	73,046
	/Admin/ManageAdmincourse.aspx	72,779
	/Admin/ManagePersonalResearch.aspx	60,741
Accessing User Metrics	Unique Users (IP addresses)	5019
	Top Geographical Locations (Singapore, Null, U.S.A, Thailand, Japan, China)	49
HTTP Status Codes	Successful Requests (200)	442,013
	Client Errors (4xx)	241,775
	Server Errors (5xx)	166,599
	Redirection	95,716
User Agent Browser (Top 5)	Chrome 109.0.0	291,193
	IE 7.0	258,632
	Null	50,115
	Chrome 114.0.0	31,336
	Chrome 108.0.0	28,418

Table 7. *Cont.*

Metric	Description	Value
Mobile Device (Top 6)	Other	852,607
	Null	50,113
	XiaoMi	11,539
	Spider	9761
	Mac	8504
	iPhone	7129
OS (Top 5)	Windows 10	526,445
	Windows XP	260,644
	Null	50,115
	Windows 7	43,346
	Android 10	11,473
Energy Consumption Metrics	Average Energy Consumption per Request (Joules)	278
	Average Energy Consumption per ALAI Request (Joules)	248
	Average Energy Consumption Idle (Joules)	203

From February 2022 to May 2024, the descriptive analysis of the student-list webpage yields vital information regarding its energy consumption, performance metrics, and usage patterns. The data indicates a substantial variability in the number of prospective and enrolled students, with noticeable increases in 2023 and decreases in 2024. The variability in question highlights the significance of the webpage and its influence on user conduct. The system facilitates a methodical arrangement of super users, department administrators, and regular users, emphasizing the application's administrative nature. The data reveals significant user engagement, with an average of 1213 daily accesses, predominantly via POST (used to send data to the server) and GET (used to request data from the server) requests. The results indicate clear enhancements in performance, as the mean response times exhibit a substantial reduction from 732.5 milliseconds in 2023 to 416.2 milliseconds in 2024. The most frequently visited web addresses, including those for registration and administrative management, underscore the fundamental features that users employ.

With more than 5000 unique IP addresses and substantial international access from Singapore, the United States, and Thailand, the user metrics indicate a diverse user base. The HTTP status codes indicate many successful requests in addition to many client and server errors, thereby indicating a conspicuous area for enhancement. According to browser and device data, Chrome is the most frequently used browser, and mobile device usage is diverse, necessitating broad compatibility. The findings of the energy consumption analysis reveal that the mean energy consumption per request is 248 watts, while idle periods consume an average of 203 watts. This energy consumption underscores the criticality of optimizing both active and idle energy usage. It is crucial for the overall efficiency and sustainability of the system to be improved, emphasizing the urgency and importance of our role. The combined insights presented above offer a thorough comprehension of the operational dynamics and motivate us to investigate possible optimization strategies for the student-list webpage.

4.6. Analysis Results

The quantitative analysis provides a detailed quantitative analysis of Agile and ALAI software development methods used for the GAISS, focusing on key metrics such as the number of accesses, average time taken, total time taken, and energy consumption for each month from 2022 to mid-2024. Notably, as illustrated in Table 8, the data results reveal

significant variations in system usage and performance, with peaks in access numbers, particularly in July and August 2022, and April 2023, reflecting periods of high demand. Average response times generally improve over the years, with 2024 showing better performance compared to previous years. This improvement is evident from the reduced average response times and total time taken for page loads, suggesting successful optimization efforts. The energy consumption comparison between Agile and ALAI frameworks highlights the latter's superior efficiency, with consistently lower energy usage across all recorded months. For instance, in April 2023, Agile methods consumed approximately 122,888,159.03 watts, while ALAI consumed significantly less at 109,626,846.90 watts. This trend of lower energy consumption with ALAI persists throughout the data, underscoring the framework's effectiveness in enhancing energy efficiency. These insights collectively indicate that the adoption of the ALAI framework not only improves performance metrics but also contributes to substantial energy savings, making it a more sustainable choice for system development.

Table 8. Monthly accesses, average response times, total time taken, and energy consumption for Agile and ALAI frameworks in GAISS from February 2022 to May 2024.

Year	Month	Accesses (Times)	URL Student List	Average Time Taken (ms)	Total Time Taken (s)	Agile Energy (watts)	ALAI Energy (watts)
2022	February	19.00	-	791.10	15.03	4178.59	3727.66
	March	118.00	-	1346.50	158.89	44,170.59	39,403.98
	April	124.00	-	181.70	22.53	6263.56	5587.64
	May	6.00	-	222.50	1.34	371.13	331.08
	June	1841.00	297.00	275.20	506.64	140,846.81	125,647.51
	July	12,861.00	3936.00	2240.10	28,809.93	8,009,159.46	7,144,861.67
	August	22,325.00	6336.00	440.80	9840.86	2,735,759.08	2,440,533.28
	September	8598.00	2345.00	851.60	7322.06	2,035,531.79	1,815,870.09
	October	4603.00	1113.00	489.80	2254.55	626,764.73	559,128.25
	November	18,700.00	5773.00	355.70	6651.59	1,849,142.02	1,649,594.32
	December	22,108.00	7713.00	373.50	8257.34	2,295,539.96	2,047,819.82
2023	January	21,935.00	8132.00	332.70	7297.77	2,028,781.31	1,809,848.08
	February	15,153.00	3632.00	295.60	4479.23	1,245,225.05	1,110,848.25
	March	30,779.00	5398.00	307.50	9464.54	2,631,142.82	2,347,206.54
	April	490,125.00	5123.00	901.90	442,043.74	122,888,159.03	109,626,846.90
	May	19,932.00	5861.00	359.30	7161.57	1,990,915.79	1,776,068.76
	June	33,608.00	11,615.00	532.50	17,896.26	4,975,160.28	4,438,272.48
	July	77,859.00	10,525.00	261.00	20,321.20	5,649,293.32	5,039,657.35
	August	30,947.00	10,844.00	823.90	25,497.23	7,088,230.86	6,323,313.86
	September	18,053.00	4774.00	658.10	11,880.68	3,302,828.85	2,946,408.47
	October	9478.00	2153.00	276.20	2617.82	727,754.96	649,220.25
	November	16,927.00	3649.00	1097.10	18,570.61	5,162,630.05	4,605,511.70
	December	15,076.00	3826.00	271.00	4085.60	1,135,795.69	1,013,227.81
2024	January	17,726.00	6222.00	207.80	3683.46	1,024,002.66	913,498.77
	February	10,353.00	3233.00	363.50	3763.32	1,046,201.71	933,302.24
	March	24,730.00	3350.00	621.10	15,359.80	4,270,025.23	3,809,231.14
	April	17,128.00	3508.00	299.80	5134.97	1,427,522.88	1,273,473.65
	May	4991.00	1722.00	650.20	3245.15	902,151.20	804,796.75
	June	-	-	-	-	-	-

The monthly accessing times of the GAISS from February 2022 to May 2024 are depicted in Figure 8. The data provides insights into notable fluctuations and patterns in the utilization of the system. The data indicates relatively stable access frequencies except for significant peaks, with moderate fluctuations in most months. However, April 2023 stands out as a period of exceptionally high demand, with nearly 500,000 accesses granted. This figure, significantly higher than any other month, underscores the urgency of the situation and the need for immediate attention. This peak starkly contrasts the access frequencies that typically hover below 50,000 in the remaining months. An additional significant, albeit diminished, height is noted in July 2022, indicating intermittent surges in activity that may be associated with academic cycles or particular occasions. The gradual rise in system utilization is reflected in the marginal overall increase in access times over the period, denoted by the dotted linear trend line. This phenomenon emphasizes the increasing dependence on GAISS to manage admissions and associated procedures, highlighting the necessity for continuous performance enhancements to accommodate growing demands efficiently. It is critical to acknowledge that these ups and downs are subject to the influence of numerous factors, including admission cycles, application deadlines, and system updates. These variables should be duly considered when devising strategies for optimizing the system.

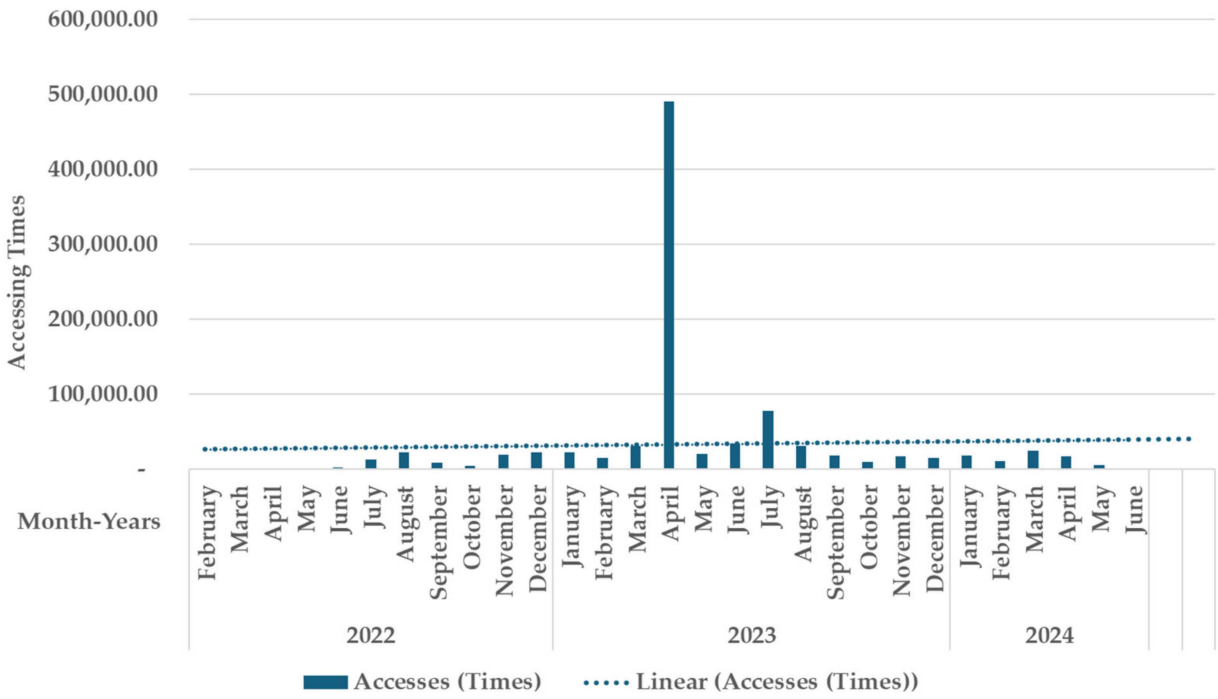

Figure 8. Monthly access frequency for the student-list webpage from February 2022 to May 2024.

A quantitative analysis of the access status distribution for the GAISS student-list webpage is presented in Figure 9. This analysis provides significant insights into the dependability of the system and the satisfaction of its users. The data provides clear evidence of the system's exceptional dependability and streamlined management of user inquiries. An estimated one hundred thousand access attempts are successful, demonstrating the system's resilience and capacity to address user requirements efficiently. This data underscores the efficacy of the GAISS in upholding a superior standard of performance and dependability, guaranteeing users a smooth and uninterrupted experience when accessing the student-list webpage.

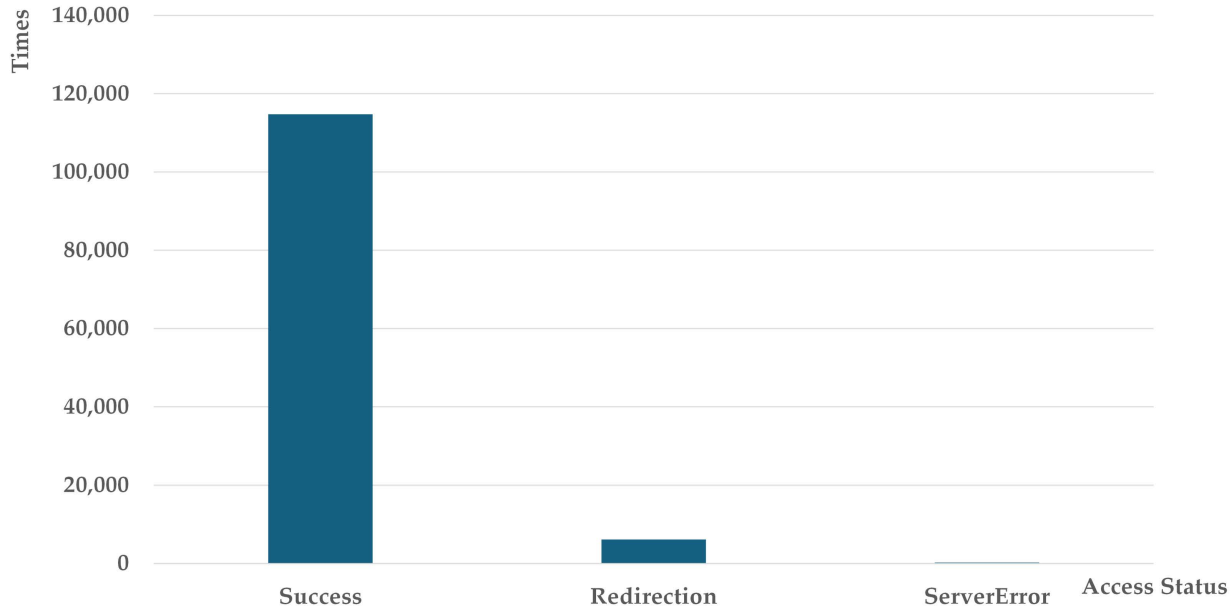

Figure 9. Distribution of access status for the GAISS student-list webpage.

The monthly access counts and average response times for the GAISS student-list webpage between February 2022 and May 2024 are quantitatively analyzed in Figure 10. The insights provided by this analysis regarding usage patterns and system performance are significant. The access counts, denoted by orange bars, exhibit a consistent upward trajectory, with significant surges observed in April 2023 and multiple months in 2022, notably July and August. These surges in activity probably coincide with critical academic or administrative periods. The average response times, denoted by the blue line, demonstrate a relatively consistent trend despite minor fluctuations. Notwithstanding the escalating volume of accesses, the mean response times consistently remain low, highlighting the system's resilience to manage heightened workloads without substantial performance deterioration. The dotted linear trend lines representing response times and access counts further underscored the trends. The upward trend in the access count line and the slight decline in the response timeline emphasize system performance and scalability enhancements. This analysis highlights the resilience and capability of GAISS to efficiently handle increasing demand while preserving prompt response times, guaranteeing a dependable user experience, even during periods of high usage.

A comprehensive analysis of the monthly energy consumption of the Agile and ALAI frameworks in GAISS from February 2022 to May 2024 is presented in Figure 11. This comparison, based on reliable data, provides significant insights into the energy efficiency of both development approaches. The bar graph, with Agile's energy consumption depicted in orange and ALAI's in green, clearly shows that ALAI consistently maintains a lower energy consumption than Agile. The dependability of these frameworks is particularly evident during periods of high activity, such as April 2023, when both experience a substantial increase in energy consumption due to increased demand on the systems. Despite this peak, ALAI sustains a comparatively modest energy consumption, demonstrating its reliability and lower energy footprint than Agile. The consistent decrease in energy consumption further underscores the dependability and efficacy of the ALAI framework in facilitating sustainable software development through optimizing resource allocation and reducing energy wastage. The data illustrates that implementing the ALAI framework can result in significant energy conservation, a critical factor in mitigating the system's overall environmental footprint and enhancing operational effectiveness.

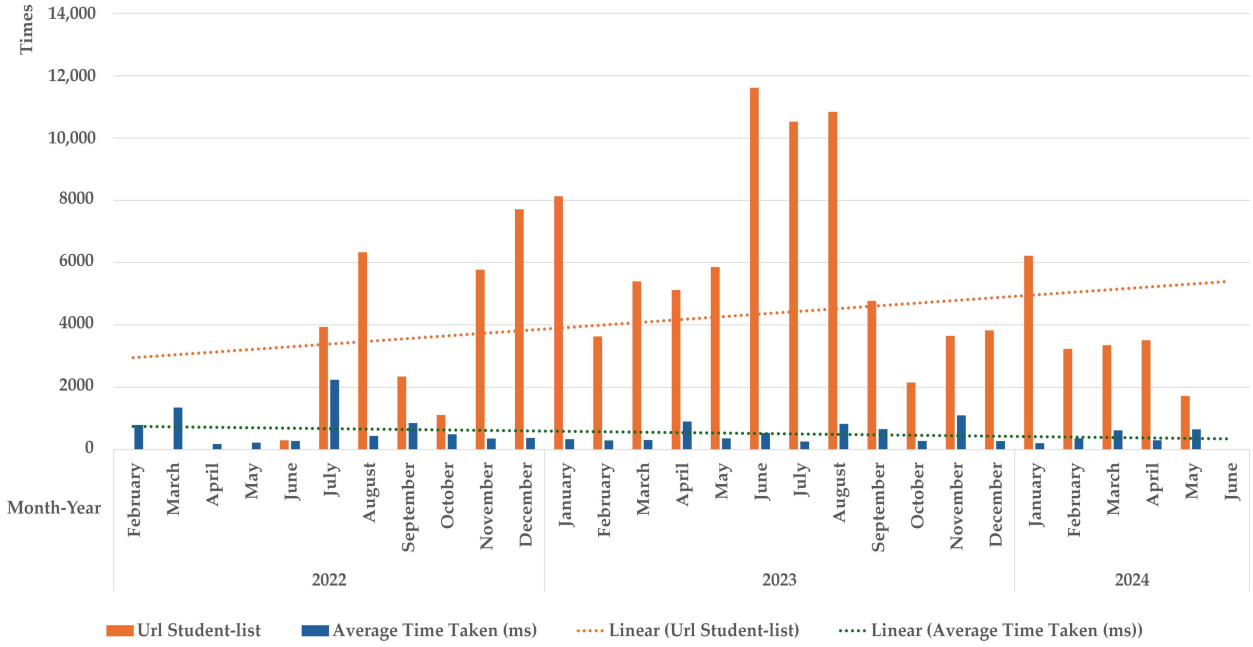

Figure 10. Monthly access counts and average response times for the GAISS student-list webpage from February 2022 to May 2024.

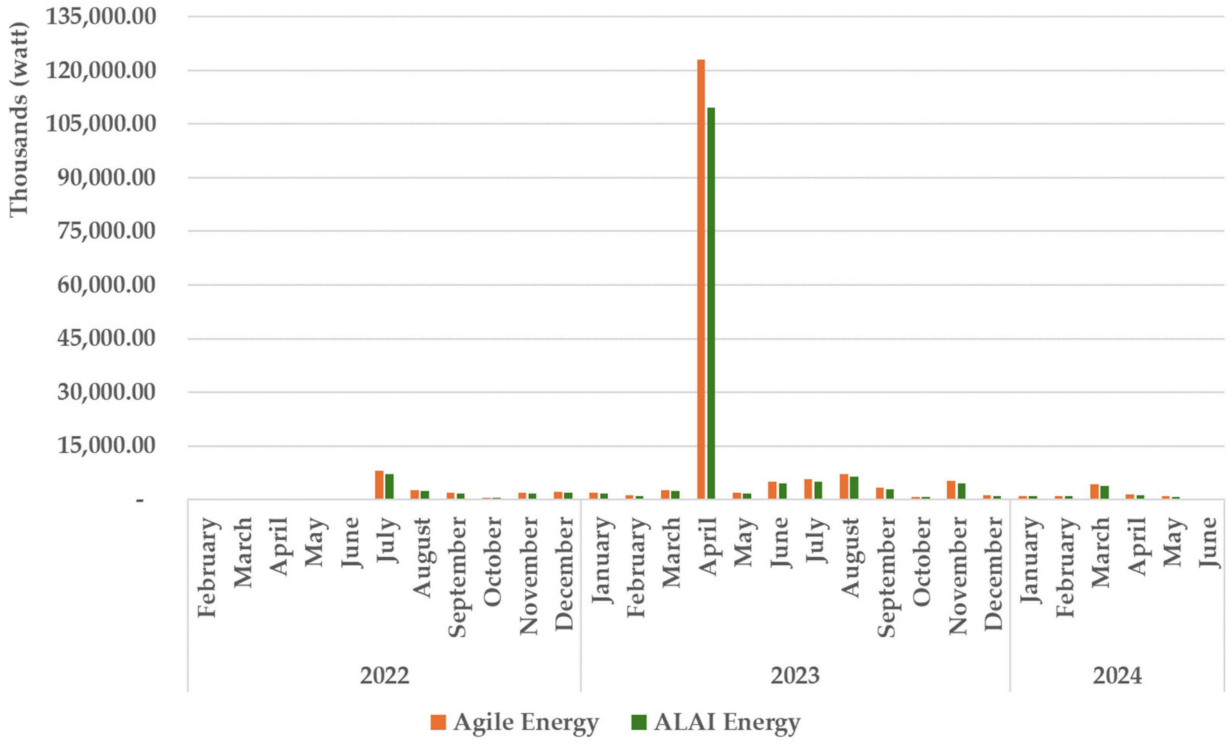

Figure 11. Comparison of monthly energy consumption for Agile and ALAI frameworks in GAISS from February 2022 to May 2024.

The evaluation of energy usage for the student-list webpage in GAISS indicates a significant enhancement in effectiveness when the ALAI methodology is implemented in contrast to the conventional Agile approach. The Agile system utilized a cumulative of 185.095 h of operation, during which it consumed 185,243.55 kW or 1000.80 kWh. The ALAI framework, in contrast, utilized 892.80 kWh or 165,253.24 kW. The energy

consumption figures indicate a substantial decline in energy usage, as ALAI exhibits an efficiency enhancement of 10.7914%. This significant decrease in energy usage not only signifies a considerable enhancement in operational effectiveness but also underscores the environmental benefits of the ALAI framework. By promoting the adoption of energy-conscious software development environments, ALAI helps mitigate the environmental consequences linked to extended system utilization, a crucial step towards sustainable software development.

5. Conclusions

The study's conclusion validates the substantial discoveries regarding the GAISS's preference for the ALAI framework compared to conventional Agile methodologies. The principal research aim was to assess the efficacy of the ALAI framework in facilitating architectural flexibility in software development and enhancing energy efficiency. The empirical data, systematically gathered between February 2022 and May 2024, presents compelling proof that the ALAI framework effectively mitigates energy consumption by 10.7914% compared to the conventional Agile methodology. In particular, the ALAI framework-based GAISS utilized 892.80 kWh of energy, while the conventional Agile methodology-based system consumed 1000.80 kWh. The significant decrease of around 108 kWh highlights the ability of the ALAI framework to maximize resource utilization and minimize environmental impact. These findings confirm the research hypothesis that system sustainability is enhanced when AOF and layered architecture are combined within Agile methodologies.

5.1. Theoretical Contributions

This study presents significant findings in sustainable software development. The ALAI framework, which has been empirically validated, offers a more pragmatic methodology, emphasizing the concrete advantages of scalability and modularity. AOF and layered architecture, which are both integrated into the ALAI framework, further enhance system modularity and flexibility. This research underscores the importance of these components in improving the effectiveness and longevity of the software development life cycle. Our research adds to the existing literature by demonstrating that substantial improvements in energy efficiency are possible without compromising system flexibility and performance.

5.2. Practical Implications

The results of this study provide practical implications for educational institutions and analogous organizations that seek to decrease their environmental impact on energy consumption and improve their overall operational effectiveness. These entities can attain significant energy conservation by implementing the ALAI framework, diminishing operational expenditures, and enhancing sustainability objectives. Significantly, the research demonstrates that implementing the ALAI framework facilitates the development of more flexible and manageable systems and produces immediate energy savings, a critical factor in ensuring long-term sustainability. It not only addresses immediate energy consumption concerns but promotes long-term sustainability by making software systems more modular, maintainable, and adaptable to future technological advancements. The potential for industry-wide shifts in software development practices is not just a possibility, but a promising reality. This encourages the design and implementation of more eco-friendly and efficient software solutions, providing a foundation for policymakers and software development teams to consider sustainability as a core criterion in their development processes, ultimately contributing to the global efforts toward reducing the carbon footprint of the IT industry.

5.3. Limitations and Future Work

However, it's important to acknowledge the limitations of this study. The findings, while significant, may have limited applicability to other industries or systems due to the

unique circumstances of GAISS. Implementing ALAI may present challenges for specific development teams due to the complexity of the process, which necessitates a particular level of expertise in layered architecture and aspect-oriented engineering. Compared to conventional Agile methodologies, ALAI's initial setup and planning time may be more demanding. The diversity of software systems across various sectors may only be partially reflected in the distinctive attributes of educational admission systems, including user load patterns and data processing demands. Therefore, the role of future studies in validating the broader practicality of the ALAI framework in diverse environments, architectures, and software platforms is crucial. These studies, which will undoubtedly make significant contributions to subsequent research endeavors, must also address any identified obstacles, such as the complexity of integration or industry-specific demands, to enhance the framework's applicability and efficacy. The importance of future research in this endeavor cannot be overstated.

This research provides robust evidence supporting the implementation of the ALAI framework for sustainable software development. The potential of ALAI to transform software development practices is evident in the substantial energy savings and enhanced system flexibility it delivers. This not only fosters optimism about a more sustainable future but also instills confidence in the framework's potential. By integrating the advanced methodologies of layered architecture and AOF, the ALAI framework facilitates the development of more flexible, efficient, and environmentally sustainable software systems, thereby promoting operational and environmental excellence. Further investigation and enhancement of this framework in subsequent studies would amplify its benefits and suitability across various contexts, reinforcing this confidence in its potential.

Author Contributions: Conceptualization, P.N., B.S. and M.R.; methodology, P.N., M.R. and B.S.; software evaluation and modeling, P.N., M.R. and B.S.; validation, P.N., M.R. and B.S.; formal analysis, P.N., M.R. and B.S.; investigation, B.S., P.N. and M.R.; resources, B.S. and M.R.; data curation, P.N., M.R. and B.S.; writing—original draft preparation, P.N., M.R. and B.S.; writing—review and editing, P.N. and M.R.; visualization, P.N., M.R. and B.S.; supervision, P.N. and M.R.; project administration, P.N., M.R. and B.S.; funding, P.N. and B.S. All authors have read and agreed to the published version of the manuscript.

Funding: This research did not receive specific grant funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The original contributions presented in the study are included in the article, further inquiries can be directed to the corresponding author.

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

References

1. Willms, P.; Brandenburg, M. Emerging trends from advanced planning to integrated business planning. *IFAC-PapersOnline* **2019**, *52*, 2620–2625. [CrossRef]
2. Rukhiran, M.; Netinant, P. A practical model from multidimensional layering: Personal finance information framework using mobile software interface operations. *J. Inf. Commun. Technol.* **2020**, *19*, 3. [CrossRef]
3. Blanco, J.Z.; Lucrédio, D. A holistic approach for cross-platform software development. *J. Syst. Softw.* **2021**, *179*, 110985. [CrossRef]
4. Bastidas Fuertes, A.; Pérez, M.; Meza, J. Transpiler-based architecture design model for back-end layers in software development. *Appl. Sci.* **2023**, *13*, 11371. [CrossRef]
5. Jánki, Z.R.; Bilicki, V. The impact of the web data access object (WebDAO) design pattern on productivity. *Computers* **2023**, *12*, 149. [CrossRef]
6. Camañes, V.; Tobajas, R.; Fernandez, A. methodology of eco-design and software development for sustainable product design. *Sustainability* **2024**, *16*, 2626. [CrossRef]
7. Oyedeji, S.; Seffah, A.; Penzenstadler, B. A catalogue supporting software sustainability design. *Sustainability* **2018**, *10*, 2296. [CrossRef]
8. Reyna, J.; Hanham, J.; Orlando, J. From e-waste to eco-wonder: Resurrecting computers for a sustainable future. *Sustainability* **2024**, *16*, 3363. [CrossRef]

9. Haputhanthrige, V.; Asghar, I.; Saleem, S.; Shamim, S. The impact of a skill-driven model on scrum teams in software projects: A catalyst for digital transformation. *Systems* **2024**, *12*, 149. [CrossRef]
10. Bein, W. Energy saving in data centers. *Electronics* **2018**, *7*, 5. [CrossRef]
11. Rukhiran, M.; Boonsong, S.; Netinant, P. Sustainable optimizing performance and energy efficiency in proof of work blockchain: A multilinear regression approach. *Sustainability* **2024**, *16*, 1519. [CrossRef]
12. Kuaban, G.S.; Gelenbe, E.; Czachórski, T.; Czekalski, P.; Tangka, J.K. Modelling of the energy depletion process and battery depletion attacks for battery-powered internet of things (IoT) devices. *Sensors* **2023**, *23*, 6183. [CrossRef]
13. Sriraman, G.; Raghunathan, S. A systems thinking approach to improve sustainability in software engineering—A grounded capability maturity framework. *Sustainability* **2023**, *15*, 8766. [CrossRef]
14. Fagarasan, C.; Cristea, C.; Cristea, M.; Popa, O.; Pisla, A. Integrating sustainability metrics into project and portfolio performance assessment in agile software development: A data-driven scoring model. *Sustainability* **2023**, *15*, 13139. [CrossRef]
15. Ciancarini, P.; Ergasheva, S.; Kholmatova, Z.; Kruglov, A.; Succi, G.; Vasquez, X.; Zuev, E. Analysis of energy consumption of software development process entities. *Electronics* **2020**, *9*, 1678. [CrossRef]
16. García-Berná, J.A.; Fernández-Alemán, J.L.; Carrillo de Gea, J.M.; Toval, A.; Mancebo, J.; Calero, C.; García, F. Energy efficiency in software: A case study on sustainability in personal health records. *J. Clean. Prod.* **2021**, *282*, 124262. [CrossRef]
17. Alrabaiah, H.A.; Medina-Medina, N. Agile Beeswax: Mobile app development process and empirical study in real environment. *Sustainability* **2021**, *13*, 1909. [CrossRef]
18. Manimegalai, R.; Sandhanam, S.; Nandhini, A.; Pandia, P. Energy efficient coding practices for sustainable software development. In Proceedings of the First International Conference on Science, Engineering and Technology Practices for Sustainable Development, Coimbatore, India, 17–18 November 2023. [CrossRef]
19. Vračar, L.M.; Stojanović, M.D.; Stanimirović, A.S.; Prijić, Z.D. Influence of encryption algorithms on power consumption in energy harvesting systems. *J. Sens.* **2019**, *2019*, 8520562. [CrossRef]
20. Strojny, J.; Krakowiak-Bal, A.; Knaga, J.; Kacorzyk, P. Energy security: A conceptual overview. *Energies* **2023**, *16*, 5042. [CrossRef]
21. Huang, S.M.; Yen, D.C.; Yan, T.J.; Yang, Y.T. An intelligent mechanism to automatically discover emerging technology trends: Exploring regulatory technology. *ACM Trans. Manag. Inf. Syst.* **2022**, *13*, 1–29. [CrossRef]
22. Ozdenizci Kose, B. Business process management approach for improving agile software process and agile maturity. *J. Softw. Evol. Process.* **2021**, *33*, e2331. [CrossRef]
23. Donca, I.-C.; Stan, O.P.; Misaros, M.; Gota, D.; Miclea, L. Method for continuous integration and deployment using a pipeline generator for agile software projects. *Sensors* **2022**, *22*, 4637. [CrossRef]
24. Erdenebat, B.; Bud, B.; Batsuren, T.; Kozsik, T. Multi-project multi-environment approach—An enhancement to existing DevOps and continuous integration and continuous deployment tools. *Computers* **2023**, *12*, 254. [CrossRef]
25. Waseem, M.; Liang, P.; Shahin, M. A systematic mapping study on microservices architecture in DevOps. *J. Syst. Softw.* **2020**, *170*, 110798. [CrossRef]
26. Hassan, H.B.; Barakat, S.A.; Sarhan, Q.I. Survey on serverless computing. *J. Cloud Comput. Adv. Syst. Appl.* **2021**, *10*, 39. [CrossRef]
27. Sadowski, C.; Zimmermann, T. *Rethinking Productivity in Software Engineering*; Apress: Berkeley, CA, USA, 2019; pp. 29–37.
28. Subramanya, R.; Sierla, S.; Vyatkin, V. From DevOps to MLOps: Overview and application to electricity market forecasting. *Appl. Sci.* **2022**, *12*, 9851. [CrossRef]
29. Karamitsos, I.; Albarhami, S.; Apostolopoulos, C. Applying DevOps practices of continuous automation for machine learning. *Information* **2020**, *11*, 363. [CrossRef]
30. Dustdar, S.; Gall, H. Architectural Concerns in Distributed and Mobile Collaborative Systems. *J. Syst. Arch.* **2003**, *49*, 521–522. [CrossRef]
31. Pashutan, M.; Abdolvand, N.; Harandi, S.R. The impact of IT resources and strategic alignment on organizational performance: The moderating role of environmental uncertainty. *Digit. Bus.* **2022**, *2*, 100026. [CrossRef]
32. Lavy, I.; Rami, R. The circumstances in which modular programming becomes the favor choice by novice programmers. *Int. J. Mod. Educ. Comput. Sci.* **2018**, *10*, 1–12. [CrossRef]
33. Rathee, A.; Chhabra, J.K. Metrics for reusability of java language components. *J. King Saud Univ.-Comput. Inf. Sci.* **2022**, *34*, 5533–5551. [CrossRef]
34. Zein, A. Implementation of service oriented architecture in mobile applications to improve system flexibility, interoperability, and scalability. *J. Inf. Syst. Technol. Eng.* **2024**, *2*, 171–174. [CrossRef]
35. Gupta, C.; Fernandez-Crehuet, J.M.; Gupta, V. Measuring impact of cloud computing and knowledge management in software development and innovation. *Systems* **2022**, *10*, 151. [CrossRef]
36. Söylemez, M.; Tekinerdogan, B.; Kolukısa Tarhan, A. Feature-driven characterization of microservice architectures: A survey of the state of the practice. *Appl. Sci.* **2022**, *12*, 4424. [CrossRef]
37. Kithulwatta, W.M.C.J.T.; Jayasena, K.P.N.; Kumara, B.T.G.S.; Rathnayaka, R.M.K.T. Integration with Docker container technologies for distributed and microservices applications: A state-of-the-art review. *Int. J. Syst. Serv.-Oriented Eng.* **2022**, *12*, 1–22. [CrossRef]
38. Mishra, A.; Otaiwi, Z. DevOps and software quality: A systematic mapping. *Comput. Sci. Rev.* **2020**, *38*, 100308. [CrossRef]
39. Netinant, P.; Elrad, T.; Fayad, M.E. A layered approach to building open aspect-oriented systems: A framework for the design of on-demand system demodularization. *Commun. ACM* **2001**, *44*, 83–85. [CrossRef]

40. Alharbi, I.M.; Alyoubi, A.A.; Altuwairiqi, M.; Ellatif, M.A. Enhance risks management of software development projects in concurrent multi-projects environment to optimize resources allocation decisions. *Int. J. Adv. Comput. Sci. Appl.* **2021**, *12*, 6. [CrossRef]
41. Akinyele, D.; Amole, A.; Olabode, E.; Olusesi, A.; Ajewole, T. Simulation and analysis approaches to microgrid systems design: Emerging trends and sustainability framework application. *Sustainability* **2021**, *13*, 11299. [CrossRef]
42. Alhammad, M.; Moreno, A. Enhancing Agile software development sustainability through the integration of user experience and gamification. In *Agile Processes in Software Engineering and Extreme Programming—Workshops*; Springer: Cham, Switzerland, 2024; pp. 12–20. [CrossRef]
43. Waheed, W.; Khodeir, L.; Fathy, F. Integrating Lean and sustainability for waste reduction in construction from the early design phase. *HBRC J.* **2024**, *20*, 337–364. [CrossRef]
44. Atadoga, A.; Umoga, U.J.; Lottu, O.A.; Sodiya, E.O. Advancing green computing: Practices, strategies, and impact in modern software development for environmental sustainability. *World J. Adv. Eng. Technol. Sci.* **2024**, *11*, 220–230. [CrossRef]
45. Ye, Y.; Barapatre, S.; Davis, M.K.; Elliston, K.O.; Davatzikos, C.; Fedorov, A.; Fillion-Robin, J.-C.; Foster, I.; Gilbertson, J.R.; Lasso, A.; et al. Open-source software sustainability models: Initial white paper from the informatics technology for cancer research sustainability and industry partnership working group. *J. Med. Internet Res.* **2021**, *23*, e20028. [CrossRef] [PubMed]
46. Kruglov, A.; Succi, G.; Vasuez, X. Incorporating energy efficiency measurement into CI\CD pipeline. In Proceedings of the 2021 2nd European Symposium on Software Engineering, Larissa, Greece, 19–21 November 2021. [CrossRef]
47. Pazienza, A.; Baselli, G.; Vinci, D.C.; Trussoni, M.V. A holistic approach to environmentally sustainable computing. *Innov. Syst. Softw. Eng.* **2024**, *20*, 1–25. [CrossRef]
48. Ghimire, D.; Charters, S. The impact of agile development practices on project outcomes. *Software* **2022**, *1*, 265–275. [CrossRef]
49. Lee, W.-T.; Chen, C.-H. Agile software development and reuse approach with scrum and software product line engineering. *Electronics* **2023**, *12*, 3291. [CrossRef]
50. Şanlıalp, İ.; Öztürk, M.M.; Yiğit, T. Energy efficiency analysis of code refactoring techniques for green and sustainable software in portable devices. *Electronics* **2022**, *11*, 442. [CrossRef]
51. Yuan, J.; Gao, Z.; Xiang, Y. Green energy consumption path selection and optimization algorithms in the era of low carbon and environmental protection digital trade. *Sustainability* **2023**, *15*, 12080. [CrossRef]
52. Ahmad, S.R.A.; Yahaya, J.; Sallehudin, H. Green software process factors: A qualitative study. *Sustainability* **2022**, *14*, 11180. [CrossRef]
53. Lis, A.; Sudolska, A.; Pietryka, I.; Kozakiewicz, A. Cloud computing and energy efficiency: Mapping the thematic structure of research. *Energies* **2020**, *13*, 4117. [CrossRef]
54. Carabaş, M.; Popescu, P.G. Energy-efficient virtualized clusters. *Future Gener. Comput. Syst.* **2017**, *74*, 151–157. [CrossRef]
55. Khalifeh, A.; Al-Adwan, A.S.; Alrousan, M.K.; Yaseen, H.; Mathani, B.; Wahsheh, F.R. exploring the nexus of sustainability and project success: A proposed framework for the software sector. *Sustainability* **2023**, *15*, 15957. [CrossRef]
56. Valmohammadi, C.; Mortaz Hejri, F. Designing a conceptual green process model in software development: A mixed method approach. *Int. J. Infor. Manag. Data Insights* **2023**, *3*, 100204. [CrossRef]
57. Mishra, A.; Mishra, D. Sustainable software engineering: Curriculum development based on ACM/IEEE guidelines. In *Software Sustainability*; Springer: Cham, Switzerland, 2021; pp. 269–285. [CrossRef]
58. Naz, R.; Khan, M.N.A. Rapid applications development techniques: A critical review. *Int. J. Softw. Eng. Appl.* **2015**, *9*, 163–176. [CrossRef]
59. Beynon-Davies, P.; Carne, C.; Mackay, H.; Tudhope, D. Rapid application development (RAD): An empirical review. *Eur. J. Inf. Syst.* **1999**, *8*, 211–223. [CrossRef]
60. Behutiye, W.; Karhapää, P.; López, L.; Burgués, X.; Martínez-Fernández, S.; Vollmer, A.M.; Rodríguez, P.; Franch, X.; Oivo, M. Management of quality requirements in agile and rapid software development: A systematic mapping study. *Inf. Softw. Technol.* **2020**, *123*, 106225. [CrossRef]
61. Marnada, P.; Raharjo, T.; Hardian, B.; Prasetyo, A. Agile project management challenge in handling scope and change: A systematic literature review. *Procedia Comput. Sci.* **2022**, *197*, 290–300. [CrossRef]
62. Mishra, A.; Alzoubi, Y.I. Structured software development versus agile software development: A comparative analysis. *Int. J. Syst. Assur. Eng. Manag.* **2023**, *14*, 1504–1522. [CrossRef]
63. Rukhiran, M.; Buaroong, S.; Netinant, P. Software development for educational information services using multilayering semantics adaptation. *Int. J. Serv. Sci. Manag. Eng. Technol.* **2022**, *13*, 1–27. [CrossRef]
64. Tu, Z. Research on the application of layered architecture in computer software development. *J. Comput. Electron. Inf. Manag.* **2023**, *11*, 34–38. [CrossRef]
65. Hocaoglu, M.F. Aspect oriented programming perspective in software agents and simulation. *Int. J. Adv. Technol.* **2017**, *8*, 3. [CrossRef]
66. Chinenyeze, S.; Liu, X.; Al-Dubai, A. An aspect-oriented model for software energy efficiency in decentralised servers. In Proceedings of the 2014 Conference ICT for Sustainability, Paris, France, 24–27 August 2014. [CrossRef]
67. Implementation of Test-Driven Development in Data Access Layer within a Business System Development. Available online: https://stis.ac.id/sipadu/pegawai/upload_jurnal/file_1517383759.pdf (accessed on 2 March 2024).

68. Mihelič, A.; Vrhovec, S.; Hovelja, T. Agile development of secure software for small and medium-sized enterprises. *Sustainability* **2023**, *15*, 801. [CrossRef]
69. Understanding Layered Software Architecture. Available online: https://systemdesignschool.io/blog/layered-software-architecture (accessed on 2 March 2024).
70. Rana, M.E.; Saleh, O.S. High Assurance Software Architecture and Design. In *System Assurances*; Elsevier: Amsterdam, The Netherlands, 2022; pp. 271–285. [CrossRef]
71. Abderlrahman, M.M.; Zhan, S.; Chong, A. A three-tier architecture visual-programming platform for building-lifecycle data management. In Proceedings of the 11th Annual Symposium on Simulation for Architecture and Urban Design, Virtual Event, 25–27 May 2020.
72. Söylemez, M.; Tekinerdogan, B.; Tarhan, A.K. Microservice reference architecture design: A multi-case study. *Softw. Pract. Exp.* **2024**, *54*, 58–84. [CrossRef]
73. Boeing, P.; Leon, M.; Nesbeth, D.; Finkelstein, A.; Barnes, C. Towards an aspect-oriented design and modelling framework for synthetic biology. *Processes* **2018**, *6*, 167. [CrossRef] [PubMed]
74. Davis, A.L. Spring AOP. In *Spring Quick Reference Guide*; Apress: Berkeley, CA, USA, 2020; pp. 33–41.
75. Mehdi Ben Hmida, M.; Ferraz Tomaz, R.; Monfort, V. Applying AOP concepts to increase web services flexibility. In Proceedings of the International Conference on Next Generation Web Services Practices, Seoul, Republic of Korea, 22–26 August 2005. [CrossRef]
76. Kumar, A.; Grover, P.S.; Kumar, R. A quantitative evaluation of aspect-oriented software quality model (AOSQUAMO). *Softw. Eng. Notes* **2009**, *34*, 1–9. [CrossRef]
77. Kumar, P. Aspect-oriented software quality model: The AOSQ model. *Adv. Comput. Int. J.* **2012**, *3*, 105–118. [CrossRef]
78. Park, D.; Kang, S.; Lee, J. Design phase analysis of software qualities using aspect-oriented programming. In Proceedings of the Seventh ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, Las Vegas, NV, USA, 19–20 June 2006. [CrossRef]
79. Zuluaga, C.A.; Aristizábal, L.M.; Rúa, S.; Franco, D.A.; Osorio, D.A.; Vásquez, R.E. development of a modular software architecture for underwater vehicles using systems engineering. *J. Mar. Sci. Eng.* **2022**, *10*, 464. [CrossRef]
80. Katal, A.; Dahiya, S.; Choudhury, T. Energy efficiency in cloud computing data centers: A survey on software technologies. *Clust. Comput.* **2023**, *26*, 1845–1875. [CrossRef] [PubMed]
81. Kravets, A.G.; Egunov, V. The software cache optimization-based method for decreasing energy consumption of computational clusters. *Energies* **2022**, *15*, 7509. [CrossRef]
82. Schaarschmidt, M.; Uelschen, M.; Pulvermüller, E.; Westerkamp, C. Framework of software design patterns for energy-aware embedded systems. In Proceedings of the 15th International Conference on Evaluation of Novel Approaches to Software Engineering, Virtual Event, 5–6 May 2020. [CrossRef]
83. Paradis, C.; Kazman, R.; Tamburri, D.A. Architectural tactics for energy efficiency: Review of the literature and research roadmap. In Proceedings of the 54th Hawaii International Conference on System Sciences, Kauai, HI, USA, 5 January 2021.
84. Avotins, A.; Nikitenko, A.; Senfelds, A.; Kikans, J.; Podgornovs, A.; Sazonovs, M. Development of analysis tools for energy efficiency increase of existing data centres. In Proceedings of the 2022 IEEE 63rd International Scientific Conference on Power and Electrical Engineering of Riga Technical University, Riga, Latvia, 10–12 October 2022. [CrossRef]
85. Khan, M.U.; Abbas, S.; Lee, S.U.-J.; Abbas, A. Measuring power consumption in mobile devices for energy sustainable app development: A comparative study and challenges. *Sustain. Comput. Inform. Syst.* **2021**, *31*, 100589. [CrossRef]
86. Chaurasia, N.; Kumar, M.; Chaudhry, R.; Verma, O.P. comprehensive survey on energy-aware server consolidation techniques in cloud computing. *J. Supercomput.* **2021**, *77*, 11682–11737. [CrossRef]
87. Ravikumar, G.; Begum, Z.; Kumar, A.S.; Kiranmai, V.; Bhavsingh, M.; Kumar, O.K. Cloud host selection using iterative particle-swarm optimization for dynamic container consolidation. *Int. J. Recent Innov. Trends Comput. Commun.* **2022**, *10*, 247–253. [CrossRef]
88. Ournani, Z.; Rouvoy, R.; Rust, P.; Penhoat, J. Tales from the code #1: The effective impact of code refactorings on software energy consumption. In Proceedings of the 16th International Conference on Software Technologies, Virtual Event, 6–8 July 2021.
89. Dorokhova, M.; Ribeiro, F.; Barbosa, A.; Viana, J.; Soares, F.; Wyrsch, N. Real-world implementation of an ICT-based platform to promote energy efficiency. *Energies* **2021**, *14*, 2416. [CrossRef]
90. Kwasek, A.; Maciaszczyk, M.; Kocot, M.; Rzepka, A.; Kocot, D.; Gąsiński, H.; Prokopowicz, D. Energy saving practices in the IT area as a factor of sustainable development of the organization: A Case Study of Poland. *Energies* **2023**, *16*, 1942. [CrossRef]
91. Testasecca, T.; Lazzaro, M.; Sarmas, E.; Stamatopoulos, S. Recent advances on data-driven services for smart energy systems optimization and pro-active management. In Proceedings of the 2023 IEEE International Workshop on Metrology for Living Environment, Milano, Italy, 29–31 May 2023. [CrossRef]
92. Netinant, P. Design adaptability for multilingual mobile application software. In Proceedings of the 24th International Conference on Software Engineering and Data Engineering, San Diego, CA, USA, 12–14 October 2015.