Computing and Informatics, Vol. 45, 2026, 261–296, doi: 10.31577/cai_2026_2_261

BRIDGING THE GAP BETWEEN SOFTWARE RE-ENGINEERING AND SUSTAINABILITY: A HOLISTIC APPROACH TO MODERN SOFTWARE SYSTEMS

Islam Zada*

Department of Software Engineering
Faculty of Computing and Information Technology
International Islamic University Islamabad
Islamabad 44000, Pakistan
e-mail: islam.zada@iiu.edu.pk

Hessa Alfraihi

Department of Information Systems
College of Computer and Information Sciences
Princess Nourah bint Abdulrahman University
Riyadh, 11671, P.O. Box 84428, Saudi Arabia
e-mail: haalfraihi@pnu.edu.sa

Sara Shahzad

Department of Computer Science, University of Peshawar
Peshawar 25000, Pakistan
e-mail: sara@uop.edu.pk

Abdullah Alsaeedi

Department of Computer Science
College of Computer Science and Engineering
Taibah University, Medina 42353, Saudi Arabia
e-mail: aasaeedi@taibahu.edu.sa

Manal Aldhayan

Department of Computer Science, College of Computer and IT
Shaqra University, Shaqra 11961, Saudi Arabia
e-mail: maldhayan@su.edu.sa

Attiya Kanwal, Safia Zaheer

Department of Bioinformatics
Faculty of Computing and Information Technology
International Islamic University Islamabad, 44000, Pakistan
e-mail: {attiya.kanwal, safia.bsbi721}@iiu.edu.pk

Hussein A. Al-Hashimi

College of Computer and Information Sciences, King Saud University
Riyadh, Saudi Arabia
e-mail: halhashimi@ksu.edu.sa

Abstract. Software Re-Engineering (SRE) and Sustainable Software Engineering (SSE) are distinct yet interrelated paradigms in modern software development. SRE focuses on restructuring and optimizing existing software systems to improve functionality, maintainability, and adaptability, while SSE emphasizes incorporating environmentally responsible practices throughout the software life cycle. This paper presents a comparative analysis of these two approaches, highlighting their foundational principles, methodologies, challenges, and benefits. It underscores the necessity of embedding green and resource-efficient practices into re-engineering activities to achieve both operational excellence and environmental sustainability. By integrating SRE with SSE, organizations can enhance software quality, reduce ecological impact, and ensure long-term viability within the evolving software ecosystem. The study further advocates for energy-efficient design, user engagement, and renewable energy integration in software infrastructure. The findings aim to guide practitioners and researchers toward developing software systems that are not only optimized for performance but also aligned with sustainability goals, thus bridging the gap between technological advancement and environmental stewardship.

Keywords: Software engineering, sustainability, re-engineering, software metrics

Mathematics Subject Classification 2010: 68N01, 68M15, 68U35

* Corresponding author

1 INTRODUCTION AND BACKGROUND

Software re-engineering and sustainable software engineering are two interrelated but different approaches to improving today's software systems. Software re-engineering mainly deals with the enhancement and change of the existing software in order to improve the functional, maintainability and performance aspects. On the other hand, sustainable software engineering practices are environmentally friendly practices in the entire process of software development, particularly the impacts of software [1]. Both approaches are important in today's software engineering practice, although they are conceptually distinct with respect to the range of goals and tasks they encompass. While software re-engineering focuses on enhancing the functionality, maintainability, and performance of existing software systems, sustainable software engineering emphasizes the incorporation of environmentally responsible practices throughout the software lifecycle, including development, deployment, operation, and maintenance [2, 3, 4].

Software Re-Engineering (SRE) could be defined as the process of modifying, improving, or transforming the existing software systems with the aim of improving their quality, functionality and maintainability as well as performance. The main purpose of this approach is to respond to new business demands, satisfy changing users' expectations, use new technologies, and fight against software ageing. While new application development aims at developing new application systems from the ground up, software re-engineering focuses on enhancing the current systems. Some of the reasons that require this process include the rate of technology change, the need to support new platforms or devices, the desire to develop in using new techniques, and the objective of enhancing the maintainability and scalability of software [1, 5, 6].

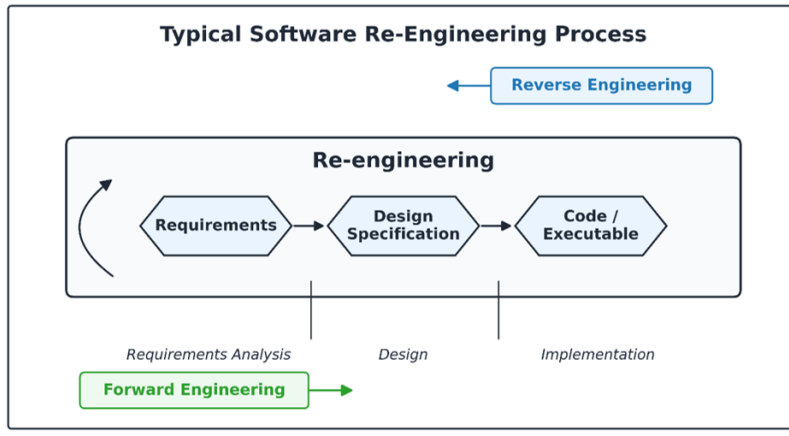


Figure 1. Typical software re-engineering process [7, 8]

The holistic approach to software re-engineering typically consists of two major phases: **reverse engineering** and **forward engineering**, as shown in Figure 1. The traditional representation of the re-engineering process is illustrated in Figure 2 [7, 8].

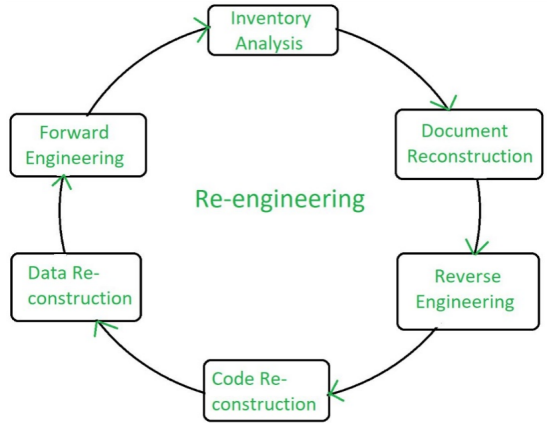


Figure 2. Classical representation of the re-engineering approach

In contrast, Sustainable Software Engineering (SSE) aims to integrate sustainability and environmental considerations into the software development process. This approach recognizes the significant environmental impact of software systems and seeks to minimize energy consumption, resource usage, and carbon emissions associated with software development, deployment, and usage. The overarching goal is to design, develop, and maintain software systems in a manner that is environmentally, socially, and economically sustainable. Key aspects of sustainable software engineering include optimizing energy efficiency, maximizing resource utilization, minimizing waste, employing green technologies, and ensuring compliance with environmental regulations and standards [9]. Figure 3 offers a representation of green and sustainable software engineering, highlighting its dimensions and overall efficacy [10, 11].

Figure 4 further outlines structured guidelines for sustainable software engineering, detailing practices across the various phases of the software development lifecycle, from requirements gathering to architecture, implementation, testing, and maintenance. These guidelines encompass approximately 20 distinct approaches, promoting sustainability without implying adherence to the waterfall model.

2 IMPORTANCE OF INTEGRATION

Sustainability of software systems can be defined as an optimization of software systems to be effective and efficient not only in delivering their intended services, but

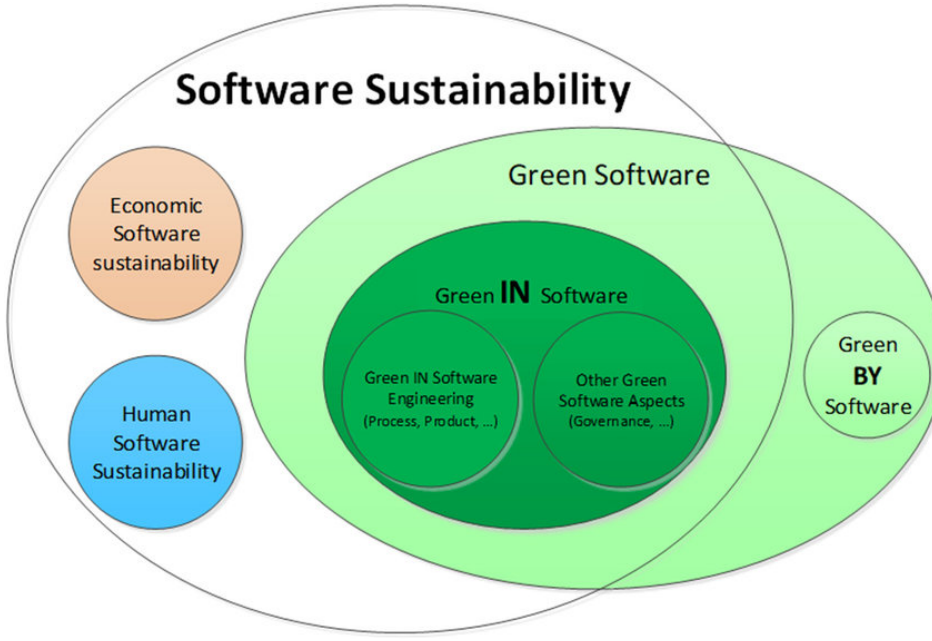


Figure 3. Associated representation of green and sustainable software engineering

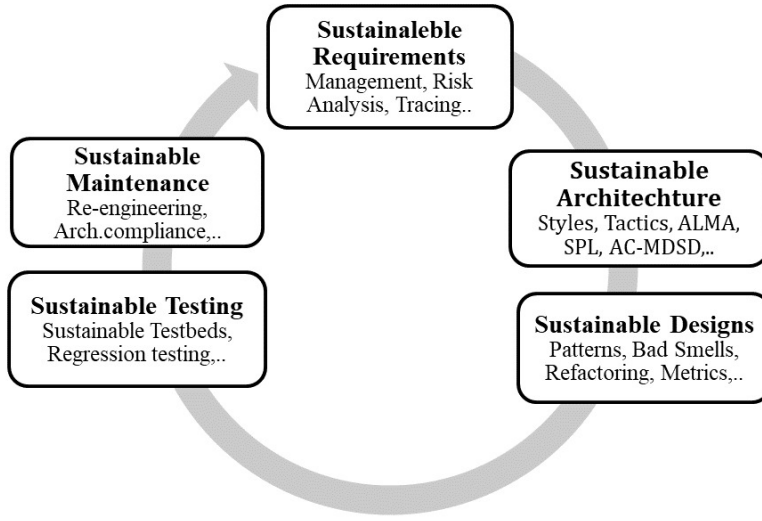


Figure 4. Structured guidelines of sustainable software engineering [12]

also in their impact on the environment. This integration is based on the knowledge of the software re-engineering and sustainable software engineering concepts and their interdependence. There are several reasons why software re-engineering and sustainable software engineering should be integrated. Over time, software systems become larger and more complex, and there is a need to enhance and update these systems with emphasis on the environmental aspect [13, 14]. Software re-engineering offers ways of enhancing existing software; sustainable software engineering makes sure that these enhancements are environmentally sustainable [1, 15, 16]. The application of sustainable practices in reengineering activities can help organizations to minimize the effects of climate change in software development and maintenance. This consists of minimizing energy use, efficient utilization of resources and the application of green technologies. Furthermore, the integration of these two strategies also guarantees the longevity of the software systems since it allows it to be continually modified to fit the ever-evolving technological, business, and environmental requirements [17, 18].

3 OBJECTIVES AND SCOPE OF SOFTWARE RE-ENGINEERING AND SUSTAINABLE SOFTWARE ENGINEERING

The objectives of software re-engineering and sustainable software engineering are distinct yet interrelated, and understanding these goals is essential for developing effective, sustainable software systems. These objectives vary depending on the specific aims and contexts of a given project.

3.1 Objectives of Software Re-Engineering

1. The primary objective of software re-engineering is therefore to improve the capabilities of the existing software systems. This includes areas of weakness, new additions and a better user interface in order to suit changing business needs and users. Another important objective is to enhance the sustainability of the software systems by refactoring and restructuring the code. This is achieved through enhancing the code readability, the extent of modularization and the reduction of code complexity so that future changes can be easily made [19, 20, 21].
2. Software re-engineering because of the fast advancement in technology seeks to transform software systems to adopt the new platforms, frameworks, and tools. This not only improves performance, compatibility and scalability but also ensures that the systems are standardized to meet industry standards and benchmarks. In addition, software re-engineering enables software systems to be responsive to the changes in the environment, such as operating systems, hardware platforms or regulations. This flexibility ensures that software is usable in a world that is constantly changing technologically.

3. In addition, another main goal of software re-engineering is to enhance system performance by enhancing algorithms, data handling, and utilization of resources. This results in more efficient systems and improves the general usability [22, 23]. Table 1 provides a comprehensive overview of these objectives.

3.2 Objectives of Sustainable Software Engineering

Sustainable software engineering seeks to align software engineering practices with broader environmental and sustainability goals. The main objective is to minimize the ecological footprint of software systems throughout their entire lifecycle, from development to deployment and maintenance [28, 29]. The specific objectives and explanations are outlined in Table 2.

Another well-defined aim of sustainable software engineering is to work towards making the principles of sustainability a part of the software development life cycle. This involves evaluation for energy and resource saving, effective and efficient usage of resources, minimizations of wastage, use of natural resources, amongst other aspects, and compliance with environmentally friendly policies. Further, actions should be taken to raise the consciousness of people about the issue and to advocate the principles of sustainable software engineering among the members of the software engineering community. In this case, awareness will be created among developers on the impact of software on the environment and ensure that next time they develop software, they also consider the impact on the environment. Sustainable software engineering also deals with the advocacy of the emergence of new and environmentally sustainable solutions when it comes to software products and technologies. This involves looking for better ways, methods, and techniques, means that will go a long way in reducing the effect of software systems on the environment. In addition, sustainable software engineering is meant to deliver economic and social values to organizations, for example, minimizing costs, improving productivity, efficiency, and energy conservation [31, 32, 33, 34].

Several studies have demonstrated that software modernization and re-engineering activities can contribute to improved energy efficiency and reduced resource consumption. Refactoring legacy code, optimizing architectural components, improving resource allocation mechanisms, and adopting energy-aware development practices have been reported to lower operational energy demands while simultaneously improving software maintainability and performance. Sustainable software engineering therefore, complements re-engineering efforts by ensuring that software evolution not only addresses technical debt and functionality requirements but also minimizes environmental impact throughout the software lifecycle [9, 13, 16].

The goals of software re-engineering and sustainable software engineering are to increase the functionality of software, its maintainability, flexibility for the future needs, and to promote innovation. Sustainable software engineering is a process

Objective	Description
Enhancement	Enhance the software's functionality, performance, and features to align with evolving company needs and user demands.
Maintainability	Revise the code to improve comprehensibility, adaptability, and maintainability, hence decreasing technical debt and enhancing the development process.
Migration	Transition the software from one platform, language, or technology to another while preserving its functionality and data.
Compatibility	Adapt the software to work effectively with new operating systems, libraries, and hardware, ensuring continued usability.
Reverse Engineering	Analyze existing software to understand its structure, behavior, and design decisions, often used when documentation is lacking or outdated.
Documentation	Create or update documentation to accurately reflect the software's architecture, design, and functionality, aiding future development and support.
Cost Reduction	Streamline and optimize the codebase to reduce resource consumption, improve efficiency, and cut down operational costs.
Bug Resolution	Identify and fix defects, vulnerabilities, and errors in the software to enhance its reliability and security.
Legacy System Transformation	Modernize a legacy software system by replacing outdated components, integrating new technologies, and aligning it with current business needs.
Performance Improvement	Analyze and optimize the software's performance, making it more responsive, scalable, and efficient.
User Experience Enhancement	Refine the software's interface and usability to provide a better overall experience for end-users.
Code Reuse	Identify opportunities to reuse existing code components in other projects, saving development time and resources.
Compliance	Ensure the software adheres to industry standards, regulations, and legal requirements, especially in regulated domains.
Risk Reduction	Mitigate risks associated with security vulnerabilities, system failures, and other potential issues through code improvement.
Business Process Alignment	Modify the software to better align with changes in the organization's business processes and workflows.

Table 1. Objectives of software re-engineering [16, 24, 25, 26, 27]

Objective	**Description**
Energy Efficiency	Develop software that minimizes energy consumption and resource usage, promoting environmentally friendly practices.
Longevity Design	Design software with a focus on longevity, ensuring it remains relevant and functional over an extended period, reducing the need for frequent replacements.
Low Environmental Impact	Minimize the carbon footprint and environmental impact of software development and operation.
Modularity and Reusability	Create modular and reusable code components to reduce duplication of effort and promote efficient resource utilization.
Minimal Resource Usage	Optimize software to use minimal system resources, such as memory and processing power, to reduce waste and energy consumption.
Lifecycle Management	Manage the entire software lifecycle, including design, development, testing, deployment, maintenance, and retirement, with sustainability in mind.
Eco-Friendly Practices	Adopt eco-friendly development practices, such as remote collaboration, digital documentation, and virtualized testing environments.
Adaptability to Change	Develop software that can easily adapt to changes in technology, business requirements, and environmental factors.
Reduced E-Waste	Build software that reduces electronic waste by prolonging the lifespan of hardware and promoting efficient resource usage.
Renewable Energy Integration	Explore opportunities to integrate with renewable energy sources or optimize energy usage based on the availability of clean energy.
Optimized Cloud Usage	Make efficient use of cloud resources to minimize energy consumption and waste in data centers.
Sustainable Design Patterns	Apply software design patterns that promote sustainability, such as the use of microservices, containerization, and serverless architecture.
Collaborative Development	Foster collaboration among developers, designers, and stakeholders to ensure a holistic and sustainable approach to software development.
Green Certification	Strive for recognized certifications or standards that attest to the software's sustainability, such as energy-efficient software labels.
Monitoring and Optimization	Continuously monitor software performance and resource usage, optimizing for efficiency and sustainability over time.

Table 2. Objectives of sustainable software engineering [18, 30, 31]

of minimizing the environmental effects of software and encouraging sustainable approach throughout the software development process.

Recent literature indicates a growing emphasis on sustainability within software engineering over the last decade. Researchers have increasingly focused on energy-efficient computing, green software development, resource optimization, lifecycle sustainability, and environmentally responsible software practices. This growing attention reflects both industrial and academic recognition of the need to reduce the environmental footprint of software systems while maintaining functionality, quality, and long-term maintainability [3, 9, 13, 16].

The domain of Software Re-Engineering and Sustainable Software Engineering is extensive and includes multiple facets of enhancing, sustaining, and guaranteeing the long-term sustainability of software systems. The main domains of both areas are below.

3.3 Scope of Software Re-Engineering

The scope of software re-engineering encompasses various facets, including [26, 35]:

- **Reverse Engineering** is the analysis of pre-existing software systems to comprehend their structure, behavior, and functionality, with the goal of producing precise and current documentation.
- **Code Restructuring** involves making modifications to the existing codebase to enhance its readability, maintainability, and efficiency while keeping its exterior behavior unchanged.
- **Migration and Porting** refer to the process of transferring a software system from one platform, language, or architecture to another without compromising its functioning.
- **Integration** refers to the process of merging various software components or systems into a coherent and unified solution.
- **Performance Enhancement:** Enhancing the performance of a software system by optimizing reaction times, minimizing resource usage, and boosting scalability.
- **Legacy System Modernization** refers to the process of upgrading or converting outdated software systems in order to align them with modern technology, standards, and user requirements.
- **Functionality Enhancement** refers to the process of incorporating more features and functionalities into an existing software system in order to cater to the changing requirements of users.
- **Documentation Enhancement:** Generating thorough and current documentation for pre-existing software systems to facilitate maintenance and future advancement.

- **Risk Mitigation** involves the identification and resolution of security vulnerabilities, coding faults, and architectural weaknesses that have the potential to cause harm to the product and its users.

3.4 Scope of Sustainable Software Engineering

The scope of sustainable software engineering includes [34, 36, 37, 38]:

- **Green Software Development:** Designing software with a focus on reducing energy consumption and minimizing environmental impact.
- **Lifecycle Management:** Ensuring the software development process accounts for environmental impacts throughout the entire lifecycle, from design to disposal.
- **Resource Efficiency:** Using software development techniques and practices that optimize the use of computing resources such as memory, processing power, and storage.
- **Reuse and Recycling:** Encouraging the reuse of code and components to reduce redundant development efforts and waste.
- **Green Software Metrics:** Developing metrics and criteria to assess the environmental impact of software systems and applications.
- **Renewable Energy Integration:** Exploring ways to integrate software with renewable energy systems to promote sustainability.
- **Environmental Compliance:** Ensuring that software systems adhere to environmental regulations and standards.
- **User Awareness and Education:** Educating users about sustainable software usage and encouraging environmentally responsible practices.
- **Continuous Improvement:** Establishing processes for ongoing evaluation and improvement of software sustainability practices.

Figure 5, a bar chart, compares different software re-engineering techniques (such as Agile, Waterfall, and Lean) against sustainability metrics such as energy consumption, resource usage, and operational costs. It also provides a comparative overview of commonly adopted software development and re-engineering approaches from a sustainability perspective. The comparison emphasizes the importance of selecting development methodologies that balance software quality, maintainability, resource utilization, and long-term sustainability objectives.

4 REASONS FOR SOFTWARE RE-ENGINEERING

This section discusses the reasons behind software re-engineering, including the necessity to improve the existing software systems. The rationales for software re-engineering can be succinctly summarized as follows:

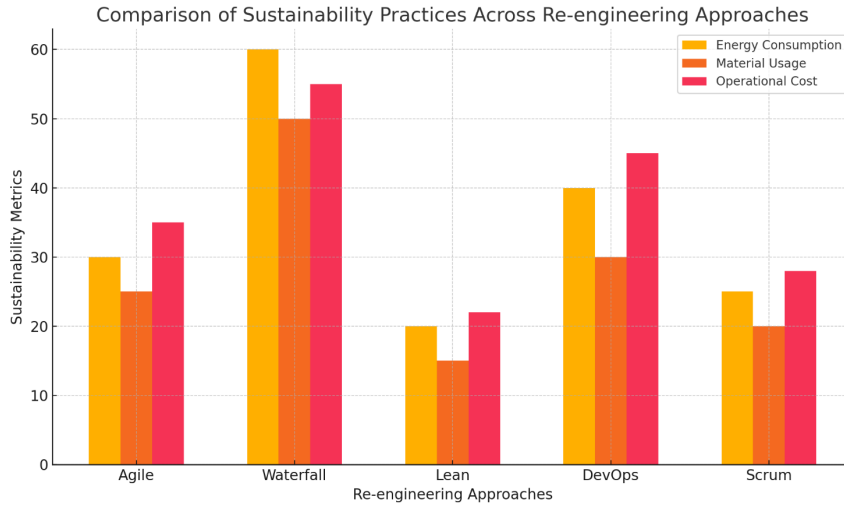


Figure 5. Comparison of sustainability practices across various software re-engineering approaches

- **Technological Obsolescence** can be described as the time when one can no longer run the software systems on new platform, on new frameworks or on new hardware, this is so because there exists a very high technological evolution rate in the market. Re-engineering is a process which makes it easier for the software to be fitted in the current technology standards.
- **Evolution of Business Requirements:** In the long run, the needs of the organizations change as well and it is possible that the software systems, which are being utilized in the organizations, may also need to be changed. We have seen from our description of re-engineering that it entails the chance to alter or modify the software to meet the needs of the organization and individuals within the organization today.
- **Software Re-Engineering** is a process that seeks to enhance the existing systems to accommodate new features or new capability to the system. It enables the expansion of the software's functionalities in a bid to help satisfy the increasing requirements of the users and the market.
- **Enhancing Maintainability:** The change and growth of the software systems becomes integrating to control, maintain, and even make modifications where necessary since most of the developed systems are poorly coded, complex and documented. Re-engineering solves these issues by simplifying the structure of codes, increasing modularity and readability of the codes and therefore making it easier to maintain and improve the software.
- **Optimization** is the act of making software perform optimally by redesigning the algorithms employed in the software to make it perform data processing

more efficiently or even in the utilization of resources. As such, the program avails improvements in its performance through response time, scalability, and overall proficiency.

- **Legacy System Modernization:** When creating systems based on outmoded technologies, certain problems may arise, such as growth, compatibility or maintenance. The act of migrating existing legacy systems to better platforms, architectures or technologies is referred to as re-engineering, and it entails the fact that the given system will run in the future [36].
- **Cost-Effectiveness:** It is often the case that re-engineering is more cost-effective than designing an all-new system. The organizations can leverage their existing software assets such as code, documentation and knowledge to reduce the time and cost of developing a new system. Table 3 shows the cost-benefit analysis of sustainable software engineering. The possible savings after five years, to show that even though the first costs may be considerably higher, the advantages of sustainable activities are tremendous. The adoption of sustainable software engineering practices may generate long-term organizational benefits that extend beyond immediate technical improvements. These benefits include reductions in operational energy consumption, lower maintenance costs, improved software longevity, and enhanced environmental sustainability. Furthermore, organizations adopting sustainable software practices can improve regulatory compliance and strengthen their reputation in terms of environmental responsibility [9, 13, 16].
- **Regulatory Compliance** means the ability to modify the software systems in an organization in order to satisfy changes in the rules or compliance standards. This is because re-engineering enables the harmonization of software with current legal or operational risks, reducing the related risks.

Benefit Area	**Expected Impact**
Energy Consumption	Reduced operational energy usage through efficient software design
Infrastructure Cost	Lower hardware and cloud resource requirements
Maintenance Cost	Improved maintainability and reduced technical debt
Software Longevity	Extended useful life of software systems
Environmental Impact	Reduced carbon footprint and resource consumption
Organizational Reputation	Improved sustainability profile and regulatory compliance

Table 3. Potential long-term benefits of sustainable software engineering

The main driver of software re-engineering pertains to improvising or enhancing currently existing software systems to cope with technological, business, and regulatory transformations. It provides a fairly cheap and practical approach towards improving software characteristics, reliability, and adaptability.

Software re-engineering as well as sustainable software engineering is very important in order to meet new demands, to increase software quality, to increase software maintainability, and for making sustainable software. Together, they cover a very important area in light of the effects of technology on organizations and the world that call for increased sustainability.

5 PROCESS AND METHODOLOGIES OF SOFTWARE RE-ENGINEERING

The process of software re-engineering entails a sequence of actions and approaches aimed at modifying and enhancing pre-existing software systems [38]. The following is a concise overview of the process and approaches involved in software re-engineering.

- **Evaluation and Strategizing:** The process commences by evaluating the current software system, discerning its merits, drawbacks, and opportunities for enhancement. An in-depth understanding of the design, functionality, and needs of the system is established. The evaluation serves as the basis for developing a re-engineering plan, which includes clear objectives, scope, resources, and timetable for the re-engineering Endeavor [39].
- **Reverse Engineering:** This stage entails examining the current software system to get knowledge about its structure, behaviour, and interdependencies. Methods such as code analysis, documentation examination, and system exploration are used to comprehend the system's components, linkages, and functionality [39].
- **Restructuring and Refactoring Process** in this structure and codebase of the software system are altered to enhance its maintainability, scalability, and modularity. This may entail restructuring the code, revamping modules, or rearranging components to boost legibility, diminish intricacy, and promote comprehension of the system.
- **Forward Engineering** involves implementing and integrating modifications or new components into an existing system after it has been restructured. This phase include the addition of new features, the enhancement of functionality, the optimization of performance, and the integration of newer technologies. The implementation adheres to industry best practices and coding standards [40].
- **Testing and Validation:** Thorough testing and validation processes are carried out to guarantee that the re-engineered software fulfils the desired functionality and quality standards. Various forms of testing, including unit test-

ing, integration testing, and system testing, are conducted to detect and address any problems or regressions that may arise during the re-engineering process [40].

- **Documentation and Knowledge Management:** Re-engineering involves updating or creating documentation to accurately reflect the changes made. This involves the act of updating and improving the documentation of a system, user guides, and technical requirements. The knowledge acquired in the re-engineered system is stored to improve future maintenance and understanding.
- **Deployment and Maintenance:** The re-engineered software is then subjected to testing and validation after which it is deployed in the production system. The assurance of constant maintenance and support tasks ensures that re-engineered software runs smoothly without any hitches. In the process of software re-engineering, different strategies and techniques can be applied. The techniques of re-engineering include model-driven re-engineering, component-based re-engineering, data-driven re-engineering, and architecture driven re-engineering. The choice of methodology depends on factors like characteristics of the software system, the specifications of the project and the tools available for use. Software re-engineering is a systematic approach to changing and improving the software systems which are already in use. Some of the activities it entails are assessment, reverse engineering, restructuring, testing, documentation, and maintenance. The goal of the process is to enhance the usability, serviceability, efficiency and compliance of the software system with new needs and advancements.

6 CHALLENGES AND LIMITATIONS

The software re-engineering, like any intricate procedure, presents its own array of obstacles and constraints. Below are few prevalent obstacles and constraints linked to software re-engineering [41, 42, 43, 44]:

- **Insufficient or Obsolete Documentation:** A major obstacle is the absence of thorough or precise documentation for the current software system. Insufficient or obsolete documentation might impede comprehension of the system's structure, capabilities, and interdependencies, hence complicating the process of re-engineering.
- **Complexity of Legacy Systems:** Legacy systems sometimes possess intricate architectures, interwoven connections, and obsolete technology, rendering them arduous to comprehend and alter. Deciphering the intricacies and updating such systems can be a laborious and demanding task.
- **Resource and Budget Constraints** refer to the limitations in terms of available resources, such as trained developers, time, and budget, that are necessary

for re-engineering activities. The re-engineering process may be affected by limited resources or budgetary constraints, which might result in compromises or incomplete improvements.

- **Stakeholder Resistance**, which may come from end-users, management, or development teams, can provide obstacles to software re-engineering. Stakeholders may exhibit resistance towards alterations to the current system or express apprehensions over the potential effects on business operations, resulting in obstacles and challenges while undertaking re-engineering initiatives.
- **Compatibility Challenges** may arise while integrating the re-engineered software system with existing systems or third-party components. Compatibility concerns might occur due to disparities in software versions, data formats, or dependencies, necessitating extra effort to guarantee seamless integration.

Software re-engineering has the potential to cause substantial changes to the current system, which can have an impact on the continuity of business operations. Operational and user experience can be affected by downtime, disruptions, or compatibility issues that may arise during the transition phase. Therefore, it is crucial to have meticulous planning and execution techniques in place.

- **Potential for the Introduction of New Problems:** If the re-engineering process is not conducted with caution, it can potentially introduce new issues or defects. Changes made during the process of restructuring, refactoring, or integration can unintentionally cause setbacks or new mistakes, necessitating comprehensive testing and validation to guarantee the stability and dependability of the re-engineered software.
- **Lack of Domain Expertise:** In certain situations, the original developers or domain experts who possess crucial knowledge about the current software system may be unavailable or inaccessible. This can present difficulties in comprehending the complexities of the system and making well-informed choices during the process of re-engineering.
- **Cultural Resistance and Change Management** are typically necessary when implementing re-engineering initiatives, as they include adopting new technologies, processes, or practices. Addressing opposition to change and effectively managing the shift within the organization can provide a considerable challenge.

Although there are hurdles and restrictions, software re-engineering can provide significant advantages in terms of enhanced functionality, maintainability, and performance. Through meticulous attention to these obstacles and the effective management of related uncertainties, organizations may adeptly navigate the process of re-engineering and fully harness the capabilities of their current software systems.

7 SIMILARITIES BETWEEN SOFTWARE RE-ENGINEERING AND SUSTAINABLE SOFTWARE ENGINEERING

Software Re-Engineering and Sustainable Software Engineering are distinct methodologies for managing and enhancing software systems, while they do exhibit certain similarities. The following are the main commonalities between these two concepts [2, 10, 21, 45, 46]:

- Both Software Re-Engineering and Sustainable Software Engineering prioritize long-term considerations. Their objective is to develop software that can dynamically adjust and develop over time to fulfil evolving demands and conditions.
- **Enhancing current systems:** Both methods entail collaborating with pre-existing software systems. Software re-engineering is the process of enhancing an existing software system to improve its maintainability, performance, or functionality. Sustainable Software Engineering prioritizes the development of new software with a focus on long-term sustainability right from the start.
- **Minimizing technical debt:** In all scenarios, the main objective is to minimize technical debt, which pertains to the buildup of suboptimal or inadequately designed code over a period of time. Resolving technical debt enhances the overall quality and maintainability of the project.
- **Environmentally conscious practices:** Sustainable Software Engineering draws influence from environmental sustainability, with the goal of reducing the software's detrimental effects on the environment. This may entail optimizing code to enhance energy efficiency or minimizing resource consumption. While Software Re-Engineering may not have a direct focus on environmental issues, enhancing the software's efficiency and performance might indirectly support sustainability.
- Both techniques can derive advantages from Agile ideas and practices, including iterative development, continuous integration, and cooperation between developers and stakeholders. Agile approaches foster flexibility and adaptation, which are crucial for achieving long-term software success.
- **User-centric approach:** Both Sustainable Software Engineering and Software Re-Engineering prioritize the demands of the end-users. Developers can ensure the longevity and worth of software by comprehending customer requirements and comments.
- Both approaches entail utilizing contemporary technologies and optimal methodologies to construct or revamp software. This guarantees that the software remains current and can fully utilize the most recent innovations.
- Both Software Re-Engineering and Sustainable Software Engineering acknowledge the significance of risk management. Effectively managing risks is crucial for ensuring long-term success, whether it involves handling old systems

during re-engineering or anticipating potential future changes during construction.

Although Software Re-Engineering and Sustainable Software Engineering have parallels, it is crucial to recognize that they have separate goals and methodologies. Software Re-Engineering is the process of enhancing and optimising existing software, whereas Sustainable Software Engineering involves developing new software with an emphasis on long-term sustainability. Although the methodologies differ, both contribute to the development of software that is durable and adaptable to changing requirements.

8 DIFFERENCES BETWEEN SOFTWARE RE-ENGINEERING AND SUSTAINABLE SOFTWARE ENGINEERING

Software Re-Engineering and Sustainable Software Engineering are two separate methodologies for the management and enhancement of software systems. The following are the primary distinctions between these two concepts [18, 36, 37, 45, 47]:

- **Emphasize the comparison between current and newly developed software:** Software re-engineering is a method that aims to enhance the quality, maintainability, performance, or functionality of existing software systems by improvement or reworking. It entails the task of dealing with outdated systems and resolving any constraints or problems they may have. On the other hand, Sustainable Software Engineering prioritizes the development of new software with sustainability as a core consideration from the beginning. The objective is to produce software that possesses intrinsic maintainability, adaptability, and environmental friendliness from the outset of the development process.
- **Chronology of factors must be considered:** Software re-engineering becomes essential where there are existing systems that have accumulated technical render or do not conform to present need. There have been past efforts aimed at improving the present software as part of this process. Sustainable Software Engineering: It is incorporated at the initial stages of the development life cycle of software products. It is a developmental strategy that involves the creation of software that could easily adapt to the new changes and additions that are on the way.
- **Purposes and aims:** Software re-engineering is therefore the process of transforming a software system to its improved versions while maintaining its functionality. The focus is on the issues that need to be addressed, such as outdated equipment, weak architecture, or low performance. Sustainable Software Engineering defines the goal to arising software system that does not fall into obsolescence within a short duration. The aim is to minimize the software impact on the environment, promote effective utilization of resources and respond to the user needs.

- **Description of activities:** This process of software structural modification is under the expansive class of re-engineering operations that include, but are not limited to code refactoring, restructuring, and technology upgrade, among others. Such efforts are made in a bid to improve the maintainability of the software, as a result, extend its lifespan. It embraces environmental concerns, energy intensity, and green practices that go into the development of software that respects notions of environmental sustainability.
- **Legacy vs. greenfield projects:** In the software re-engineering process, one has to change and enhance the systems that are already implemented and, possibly, outdated; they often have their own code and, sometimes, a database. Sustainable software engineering mainly applies to new project situations where developers can make informed decisions on the sustainable aspects of engineering.
- **Software re-engineering** is primarily a way of enhancing efficiency and the improved efficiency sometimes may have the ripple effect of reducing the environmental impact of the software. Sustainable Software Engineering: The environment is a principal aspect of sustainability when practicing software engineering. This includes practices like specifying efficient algorithms for energy use, utilization of resources and enforcing environmentally conscious practices.
- **Software Re-Engineering** is the process of enhancing existing software systems, typically legacy systems, whereas Sustainable Software Engineering is concerned with developing new software with sustainability as a primary consideration from the beginning. Both techniques have unique goals and tactics, but their ultimate objective is to develop software that can adjust and maintain its value over an extended period.

9 POTENTIAL SYNERGIES AND INTEGRATION OPPORTUNITIES

The combination of Software Re-Engineering and Sustainable Software Engineering can result in several synergies and opportunities that improve the overall quality and sustainability of software systems. Below are few possible domains for integration [42, 48, 49, 50, 51]:

- **Legacy system modernization:** Software Re-Engineering is one of the ways for achieving modernization of a legacy system. This approach proves useful in the process of modifying and refining out-of-date systems by changing their technical debt and making them more maintainable. Here it is proposed that by applying Sustainable Software Engineering principles like energy efficiency and reduced resource utilization, we can have the modernized green system.
- **The greenfield sustainable development** simply refers to the integration of the principles of sustainable software engineering in green field software develop-

ment projects at inception. However, there might be situations, in which some of the components or basic modules could indeed get benefit from re-engineering rather than to start the process of development anew. Thus, the use of such strategies can be viewed as an opportunity to create a stable background and at the same time transform and develop existing structures.

- **Code reworking**, often used in software re-engineering is the energy-efficient code reworking for the purpose of improving the quality and maintainability of the code. There are SSE concepts which, when applied by the developers, allow them to design and control the code in such a way that the consumption of resources is minimized, and the code's energy consumption is optimized.
- **Involvement of relevant parties and measurement of sustainability indicators:** Implementing Sustainable Software Engineering techniques may entail including stakeholders in the software development process to comprehend their sustainability expectations and goals. This facilitates the creation of software that is in line with sustainability objectives. In addition, the establishment and quantification of sustainability indicators during the re-engineering procedure can facilitate the monitoring of advancements and the identification of potential areas for enhancement.

Therefore, when combining Software Re-Engineering with Sustainable Software Engineering, organizations may come up with software systems that solve the problem of technical debt and add value while serving as role models for the environment. Such approaches show the importance of focusing on both short-term and long-term consequences so as to design software applications that are effective and have low impact on the environment.

10 CHALLENGES AND OPPORTUNITIES

Table 4 outlines the challenges and opportunities associated with Software Re-Engineering and Sustainable Software Engineering [15, 48, 52, 53, 54, 55], Tables 5 and 6 show the opportunities in software re-engineering and challenges in sustainable software engineering, respectively. Table 7 tells about the opportunities in sustainable software engineering.

By acknowledging and resolving these obstacles and capitalizing on the possibilities, organizations can develop software that not only fulfils their functional needs but also conforms to sustainability objectives, so benefiting the environment and promoting long-term expansion and achievement.

11 COMBINED BENEFITS AND IMPACTS

Benefits and impacts of combining Software Re-Engineering and Sustainable Software Engineering:

Challenge	**Explaination**
Legacy System Complexity	Dealing with complex legacy systems can be challenging due to outdated technologies, poor documentation, and accumulated technical debt.
Understanding Legacy Code	Gaining a thorough understanding of the existing codebase, especially in the absence of comprehensive documentation, can be time consuming and difficult.
Balancing Maintenance and Modernization	Striking a balance between maintaining existing functionality and modernizing the system to meet current requirements can be a delicate task.
Risk of Unintended Consequences	Making changes to legacy systems can introduce unforeseen bugs or compatibility issues that may have significant implications for the application.
Resistance to Change	Team members or stakeholders may resist changes to the existing system, fearing disruptions or the unfamiliarity of new technologies.
Time and Resource Constraints	Software Re-Engineering projects may require significant time, effort, and resources, leading to potential budgetary and scheduling challenges.

Table 4. Challenges along with their explaination in software re-engineering

Opportunity	**Explaination**
Performance Improvement	Re-engineering presents an opportunity to enhance system performance through code optimization, algorithm improvements, and better resource management.
Enhanced Security	Legacy systems often lack modern security measures. Re-engineering allows the integration of robust security practices to protect against potential threats.
Adopting New Technologies	Re-engineering facilitates the adoption of newer, more efficient technologies, improving the system's capabilities and supporting future growth.
Modularization and Componentization	Breaking down monolithic systems into modular components enhances maintainability and reusability, leading to easier future enhancements.
Improved Maintainability	By addressing technical debt and improving code quality, maintainability is enhanced, reducing the time and effort needed for future updates and fixes.
Alignment with Current Business Needs	Re-engineering enables alignment of the software with the organization's current business objectives and changing market demands.

Table 5. Opportunities in software re-engineering

Challenge	Explaination
Lack of Awareness and Education	The concept of sustainable software engineering may be relatively new to some developers, leading to a lack of awareness and understanding of its importance.
Balancing Sustainability and Functionality	Integrating sustainability measures without compromising essential functionality requires careful planning and decision-making.
Measuring and Quantifying Sustainability	Defining appropriate metrics and methods to measure the environmental impact of software can be challenging, making it difficult to assess improvements.
Interdisciplinary Collaboration	Sustainable software engineering requires collaboration between software developers and sustainability experts, which may introduce communication challenges.
Continuous Monitoring and Adaptation	Sustainability is an ongoing process that requires continuous monitoring and adaptation to changing environmental factors and technology advancements.
Perceived Cost Increase	There might be concerns that implementing sustainability practices could increase development costs, especially if they are not well understood.

Table 6. Challenges in sustainable software engineering

Opportunity	Explaination
Positive Environmental Impact	Sustainable software engineering offers the opportunity to develop software that positively contributes to reducing carbon footprints and environmental impact.
Competitive Advantage	Organizations can gain a competitive edge by showcasing their commitment to sustainability, attracting environmentally conscious customers and stakeholders.
Innovation and Market Opportunities	Embracing sustainability can lead to new market opportunities, as more businesses and consumers seek eco-friendly products and services.
Collaboration for Sustainability	Sustainable software engineering encourages interdisciplinary collaboration, fostering relationships with sustainability experts and organizations.
Enhanced User Perception	Users often prefer eco-friendly products, and software designed with sustainability in mind can enhance user perception and satisfaction.
Future-Proofing and Adaptability	Sustainable software engineering emphasizes adaptability and future-proofing, ensuring the software can accommodate changes in environmental regulations.

Table 7. Opportunities in sustainable software engineering

- **Improved Software Performance:** Through software re-engineering methodologies, it is possible to enhance legacy systems and eliminate problems that lead to slow processing. The use of sustainable software engineering principles leads to the creation of a system that is energy efficient and thus reduces energy consumption and energy wastage. This in turn results in improved performance of the software in question.
- **Program Duration** can be extended by reengineering the program which is a process of redesigning the program which has been in existence. Sustainable software engineering practices consider the environmental aspects, which in turn makes the software more relevant in the future when there will be new needs and regulations.
- As for the two techniques, it is possible to note that they may lead to cost savings. Software re-engineering can reduce the maintenance cost and increase the productivity, whereas sustainable software engineering reduces the resource utilization, thereby reducing the energy and infrastructure cost.
- **Enhanced User Satisfaction:** Software re-engineering improves the user experience through the improvement of the system stability, problem solving, and addition of new features. The organization's commitment to environmental sustainability in software engineering practices in the user applications is a sure way of enhancing the user loyalty and trust.
- **Enhanced Maintainability:** Software re-engineering makes the code easier to understand and maintain through the simplification of the code. SSD is the practice that encourages the reuse of the code and the efficient use of the resources because the future upgrades and maintenance will be easier.
- **Green Software Engineering Practices** can reduce energy consumption, green house gas emissions and waste, thus making software development more environmentally friendly.
- In this way, the application of sustainable software engineering practices proves that an organization is environmentally conscious and compliant with environmental requirements and social expectations.
- **Future Proofing** is the process of software re-engineering and application of sustainable software engineering to increase the ability of software systems to accommodate future changes and evolution. This makes it possible for the software to stay useful for a longer duration in the market as compared to others.
- **Innovation Facilitation** involves the process of updating and restructuring the systems in order to lay down a solid foundation for the introduction of new technologies and innovations. Sustainable software engineering is the process of developing software sustainably and being environmentally friendly in the development of software.
- **Engineering Advantage:** This paper established that organizations that implement software re-engineering and sustainable software engineering practices

are able to offer better software products and services that are more reliable, efficient and eco-friendly.

In general, the combination of software re-engineering with sustainable software engineering practices result in significant benefits for enterprises, consumers, and the environment. It increases the effectiveness of the program, extends the time for which it will remain relevant, reduces costs, improves the satisfaction of users, and also demonstrates social and environmental relevance. Thus, seeking better and more efficient software solutions, organizations can thrive in the world that is becoming increasingly technological while reducing their negative impact on the environment and contributing to the development of a less harmful future.

11.1 Best Practices in Software Re-Engineering

Software re-engineering is an important process that involves the process of refactoring, redesigning or enhancing of the existing software systems to meet new requirements, conform to the current standards and improve the overall maintainability and reliability. The following paper aims at identifying that the success of a software re-engineering project depends on the use of proper techniques that ensure the achievement of a structured and systematic approach to the project. In this document, we give a comprehensive and step-by-step guide on how to perform software re-engineering using the most efficient techniques. These techniques are conveniently presented in Table 8 in order to enhance easy and rapid reference.

When followed to the latter, the objectives of software re-engineering project can be met, and various risks and costs of maintenance and sustenance can be avoided. The following practices, when implemented in a systematic approach, may result to positive software re-engineering results.

11.2 Best Practices in Sustainable Software Engineering

Sustainable Software Engineering (SSE) is a discipline focused on developing software systems that meet current user needs while considering their long-term environmental and societal impact. Green software aims to minimize environmental impact, efficiently utilize resources, and reduce the overall cost of ownership. To achieve these objectives, it is essential to follow established guidelines that support the development and management of sustainable software solutions. Table 9 summarizes the best practices presented in this paper related to Sustainable Software Engineering [56, 57, 58].

Different sustainable software engineering practices contribute to environmental sustainability through distinct mechanisms. Code optimization reduces unnecessary computational effort and energy consumption, while data-center and server optimization improve infrastructure efficiency. Similarly, reducing hardware depen-

Best Practice	Description
Understand the Legacy System	Begin by thoroughly understanding the existing system. This includes studying the codebase, architecture, and documentation to identify the system's strengths, weaknesses, and current functionalities.
Define Clear Objectives	Clearly define the goals and objectives of the re-engineering project. Whether it is enhancing functionality, improving performance, or migrating to a new platform, having well-defined objectives is crucial.
Involve Stakeholders	Engage stakeholders, including end-users, business analysts, and development teams, to understand their needs and expectations. Regular communication and feedback loops are essential.
Comprehensive Documentation	Maintain detailed and up-to-date documentation throughout the re-engineering process. This includes architectural diagrams, data models, user manuals, and code comments.
Version Control	Implement robust version control using tools like Git. This ensures that code changes are tracked, and rollbacks are possible in case of issues.
Use of Modern Technologies	Utilize contemporary development tools, frameworks, and languages to align the re-engineered system with current industry standards and best practices.
Testing and Quality Assurance	Thoroughly test the re-engineered system to identify and fix issues. This includes unit testing, integration testing, and user acceptance testing. Implement quality assurance processes.
Iterative Approach	Adopt an iterative development process. This allows for incremental changes, ensuring that the system remains functional at all stages and reducing project risks.
Refactoring	Embrace code refactoring to improve the codebase's maintainability and readability. Eliminate redundancy, improve code structure, and apply coding standards.
Security Considerations	Prioritize security by identifying and addressing vulnerabilities in the legacy system. Implement modern security protocols and practices.
User Training and Support	Provide training to end-users and support teams on the re-engineered system. Ensure that users are comfortable with the changes and can maximize the system's benefits.
Performance Optimization	Optimize system performance through load testing, profiling, and identifying bottlenecks. Implement performance enhancements as needed.
Data Migration	Plan for the seamless migration of data from the old system to the re-engineered one. Verify data integrity and consistency.
User Acceptance Testing	Engage users in acceptance testing to ensure that the re-engineered system meets their requirements and expectations. Make adjustments based on user feedback.
Monitoring and Maintenance	Implement continuous monitoring and maintenance processes to keep the re-engineered system running smoothly. Address issues promptly and proactively.
Post-Deployment Evaluation	After deployment, conduct a post-implementation review to evaluate the re-engineering project's success and gather lessons learned for future projects.

Table 8. Best practices in Software Re-Engineering

dence minimizes electronic waste and resource consumption. The combined application of these practices can contribute to reducing the environmental footprint of software systems while maintaining acceptable levels of performance and reliability [9, 13, 55, 56, 57]. Sustainable Software Engineering practices and their environmental contributions are articulated in Table 10.

However, following the best practices laid down in the development of software, teams can design a perfect software that will fit the needs of users, without surpassing the environmental effects, overuse of resources and continuous expenses in the future. Sustainable Software Engineering helps to solve the environmental issues of the present, and these practices are the key to a less wasteful and cheaper software production.

Best Practice	Description
Requirements Prioritization	Start by prioritizing sustainability requirements alongside functional requirements. Ensure sustainability goals like reduced energy consumption are integrated into development.
Design for Efficiency	Develop software focusing on efficient resource utilization and performance by optimizing algorithms and reducing computational overhead.
Energy-Aware Development	Incorporate energy-awareness into design using tools and metrics to measure and reduce energy consumption.
Resource Minimization	Minimize memory, CPU, and storage usage using efficient data structures and algorithms.
Modular and Scalable Design	Use modular, scalable design to reduce rework and support easy adaptation to changing requirements.
Use of Sustainable Technologies	Adopt eco-friendly technologies such as energy-efficient servers and low-impact cloud services.
Code Optimization	Optimize code to reduce execution time and redundancy while improving maintainability.
Eco-Friendly Hosting	Host applications on renewable-energy-powered data centers to reduce carbon footprint.
Continuous Monitoring	Continuously monitor resource usage and energy consumption to identify inefficiencies.
Energy-Efficient Algorithms	Select algorithms based on energy efficiency in addition to computational complexity.
User Education	Educate users on sustainable usage practices and energy-efficient configurations.
Sustainability Metrics	Define and track metrics such as energy consumption and carbon footprint for evaluation.
Documentation and Training	Maintain documentation and train developers on sustainable software engineering practices.
Environmental Compliance	Ensure compliance with environmental regulations and sustainability standards.
Lifecycle Management	Plan software lifecycle responsibly, including end-of-life disposal of systems and data.
Community Engagement	Engage with sustainability communities and contribute to open-source green software initiatives.

Table 9. Best practices in Sustainable Software Engineering

11.3 Recommendations for Integrated Approaches

Recommendations for Integrated Approaches in Software Re-Engineering and Sustainable Software Engineering are discussed in Table 11. By adopting these recommendations, organizations can ensure that their software remains efficient, cost-effective, and environmentally responsible throughout its lifecycle. These practices promote a holistic and responsible approach to software development, aligning technology with sustainability goals.

Sustainable Practice	**Environmental Contribution**
Code Efficiency	Reduces computational workload and energy consumption
Data Center Optimization	Improves energy utilization and reduces operational emissions
Server Optimization	Enhances resource utilization and decreases power requirements
Hardware Reduction	Minimizes electronic waste and material consumption
Resource-Aware Programming	Improves memory and processor efficiency
Renewable Energy Integration	Reduces dependence on fossil-fuel-based energy sources

Table 10. Sustainable Software Engineering practices and their environmental contributions

These recommendations provide a structured approach to integrating sustainability into software re-engineering, ensuring that software remains efficient, cost-effective, and environmentally responsible throughout its lifecycle.

12 CONCLUSION AND FUTURE DIRECTIONS

Nowadays, in a quickly developing environment of software development, it is indispensable to combine SRE with SSE. This integration makes certain that the enhancements in the software satisfy the demands of the users and at the same time are environmentally sound. The effectiveness of using both approaches is that it helps to have long-term solutions in terms of protecting the environment from the effects of the software systems and ensuring maximization of resource utilization. Preferential features include user engagement, energy-efficient designing and the maximum use of renewable energy in the coding of such strategies. This way, synergy is created, and various organizations can develop proper software systems that would be optimized for efficiency and flexibility regarding further advances in IT and surrounding conditions. This comparative analysis also emphasizes that achieving software efficiency and software sustainability are complementary goals for software engineering to consider, given the role that such practices must play in today's software development.

Subsequent research should aim towards constructing advanced quantitative indexes and approaches to estimate the extent of software's effect on the environment, strengthening the collaboration between computer science software designers and sustainability professionals, as well as defining new innovative economic opportunities in line with sustainability. Thus, dynamism in environmental factors and technology requires the continuity of monitoring and exerting change to achieve sustainability in software engineering. Studying on how renewable energy systems can be incorporated into software as infrastructure and data centers

Recommendation	Description
Prioritize Sustainability Objectives	Make sustainability objectives a core focus from project inception, aligning them with re-engineering goals.
Lifecycle Assessment	Conduct a comprehensive lifecycle assessment to evaluate environmental impact throughout the software lifecycle.
Continuous Monitoring	Implement continuous monitoring of performance and sustainability metrics, including energy consumption.
Energy-Efficient Architecture	Design architectures that optimize resource usage, hardware utilization, and energy efficiency.
Modular and Scalable Design	Develop modular systems to support future updates and reduce resource-intensive rework.
Sustainable Coding Practices	Apply coding standards that improve efficiency, maintainability, and resource optimization.
User Education and Engagement	Educate stakeholders on sustainable usage and encourage responsible software practices.
Green Hosting	Host systems on renewable-energy-powered data centers to reduce environmental impact.
Sustainability Documentation	Maintain documentation of sustainability practices and provide training for awareness and adoption.
User Feedback Integration	Integrate feedback loops to align system evolution with sustainability and functional requirements.
Resource Optimization	Continuously optimize system resources to reduce energy consumption and computational load.
Compliance with Standards	Ensure adherence to environmental regulations, standards, and sustainability certifications.
Collaboration with Sustainability Community	Engage with sustainability communities and contribute to open-source green initiatives.
Efficient Data Management	Optimize data storage, transfer, and processing to reduce resource usage.
Integration of Renewable Energy	Explore integration of renewable energy sources into software infrastructure and deployment environments.

Table 11. Recommendations for integrated approaches

should also be pursued to expand on the carbon footprint issue of software systems.

Further, creating new methods and techniques to create automated measures for the identification and improvement of sustainable selves at different phases of the software development cycle may yield new, valuable contributions to this area of research. Momentum with the sustainability community and compliance with standard ecological policies are also pointed to as strategically important areas to boost the direction of sustainable software engineering. It is also important to look for new business models that can be less harmful to the environment, and at the same time can be mostly profitable as well, since this may also provide an insight into

how more businesses of the same industry can cooperate and adapt to sustainable business models.

Data Availability Statement: All data generated or analyzed during this study are included in this article. No additional data is available.

Competing Interests Declaration: The authors declare that they have no competing interests.

Author Contribution: Dr. Islam Zada and **Dr. Attiya Kanwal** conceptualized the research study, provided the overall direction, and contributed significantly to the manuscript writing. **Dr. Hessa Alfraihi** participated in the formulation of the research objectives and provided critical insights into sustainability. **Dr. Sara Shahzad** and **Hussein Al-Hashimi** were responsible for the literature review and assisted in drafting the introduction and background sections. **Dr. Abdullah Alsaeedi** contributed to the methodology and data analysis. **Dr. Manal Aldahyan** and **Ms. Safia Zaheer** collaborated on data collection and the organization of the results section. Also, all authors reviewed and approved the final manuscript before submission.

Acknowledgements

The authors would like to thank and acknowledge that the project is funded by the Princess Nourah bint Abdulrahman University Researchers Supporting Project ID PNURSP2026R411, Princess Nourah bint Abdulrahman University, Riyadh, Saudi Arabia, Shaqra University Shaqra, Saudi Arabia, International Islamic University Islamabad, University of Peshawar, Pakistan, King Saud University, Riyadh, Saudi Arabia, and Taibah University Medina, Saudi Arabia.

REFERENCES

[1] Muzammul, M.: Model Driven Re-Engineering with the Fields of Re-Structuring: Software Quality Assurance Theory. International Journal of Scientific and Research Publications, Vol. 8, 2018, No. 6, doi: 10.29322/IJSRP.8.6.2018.p7861.

[2] Bhavsar, K.—Shah, V.—Gopalan, S.: Scrum: An Agile Process Reengineering in Software Engineering. International Journal of Innovative Technology and Exploring Engineering (IJITEE), Vol. 9, 2020, No. 3, pp. 840–848, doi: 10.35940/ijitee.C8545.019320.

[3] Kehrer, T.—Penzenstadler, B.: An Exploration of Sustainability Thinking in Research Software Engineering. In: Chitchyan, R., Penzenstadler, B., Venter, C. C. (Eds.): Proceedings of the 7th International Workshop on Requirements Engineering for Sustainable Systems (RE4SuSy 2018) Co-Located with the 26th International Conference on Requirements Engineering (RE 2018). CEUR Workshop Proceedings, Vol. 2223, 2018, pp. 34–43, https://ceur-ws.org/Vol-2223/paper5.pdf.

[4] Grüning, B.—Dale, R.—Sjödin, A.—Chapman, B. A.—Rowe, J.—Tomkins-Tinch, C. H.—Valieris, R.—Köster, J.—The Bioconda Team: Bioconda: Sustainable and Comprehensive Software Distribution for the Life Sciences. Nature Methods, Vol. 15, 2018, No. 7, pp. 475–476, doi: 10.1038/s41592-018-0046-7.

[5] Kaur, U.—Singh, G.: A Review on Software Maintenance Issues and How to Reduce Maintenance Efforts. International Journal of Computer Applications, Vol. 118, 2015, No. 1, pp. 6–11, doi: 10.5120/20707-3021.

[6] Ali, M.—Hussain, S.—Ashraf, M.—Paracha, M. K.: Addressing Software Related Issues on Legacy Systems – A Review. International Journal of Scientific & Technology Research, Vol. 9, 2020, No. 3, pp. 3738–3742.

[7] Majthoub, M.—Qutqut, M. H.—Odeh, Y.: Software Re-Engineering: An Overview. 2018 8th International Conference on Computer Science and Information Technology (CSIT), IEEE, 2018, pp. 266–270, doi: 10.1109/CSIT.2018.8486173.

[8] Assunção, W. K. G.—Lopez-Herrejon, R. E.—Linsbauer, L.—Vergilio, S. R.—Egyed, A.: Reengineering Legacy Applications into Software Product Lines: A Systematic Mapping. Empirical Software Engineering, Vol. 22, 2017, No. 6, pp. 2972–3016, doi: 10.1007/s10664-017-9499-z.

[9] Murugesan, S.: Harnessing Green IT: Principles and Practices. IT Professional, Vol. 10, 2008, No. 1, pp. 24–33, doi: 10.1109/MITP.2008.10.

[10] Şanlialp, İ.—Öztürk, M. M.—Yiğit, T.: Energy Efficiency Analysis of Code Refactoring Techniques for Green and Sustainable Software in Portable Devices. Electronics, Vol. 11, 2022, No. 3, Art. No. 442, doi: 10.3390/electronics11030442.

[11] Rashid, N.—Khan, S. U.—Khan, H. U.—Ilyas, M.: Green-Agile Maturity Model: An Evaluation Framework for Global Software Development Vendors. IEEE Access, Vol. 9, 2021, pp. 71868–71886, doi: 10.1109/ACCESS.2021.3079194.

[12] Koziolek, H.—Weiss, R.—Durdik, Z.—Stammel, J.—Krogmann, K.: Towards Software Sustainability Guidelines for Long-Living Industrial Systems. Software Engineering 2011 – Workshopband, Gesellschaft für Informatik e.V., 2011, pp. 47–58.

[13] Dittrich, Y.: Software Engineering Beyond the Project – Sustaining Software Ecosystems. Information and Software Technology, Vol. 56, 2014, No. 11, pp. 1436–1456, doi: 10.1016/j.infsof.2014.02.012.

[14] Palacin-Silva, M. V.—Seffah, A.—Porras, J.: Infusing Sustainability into Software Engineering Education: Lessons Learned from Capstone Projects. Journal of Cleaner Production, Vol. 172, 2018, pp. 4338–4347, doi: 10.1016/j.jclepro.2017.06.078.

[15] Lee, S. U.—Fernando, N.—Lee, K.—Schneider, J. G.: A Survey of Energy Concerns for Software Engineering. Journal of Systems and Software, Vol. 210, 2024, Art. No. 111944, doi: 10.1016/j.jss.2023.111944.

[16] Napier, N. P.—Kim, J.—Mathiassen, L.: Software Process Re-Engineering: A Model and Its Application to an Industrial Case Study. Software Process: Improvement and Practice, Vol. 13, 2008, No. 5, pp. 451–471, doi: 10.1002/spip.390.

[17] Bradley, P.—Browne, J.—Jackson, S.—Jagdev, H.: Business Process Re-

Engineering (BPR) – A Study of the Software Tools Currently Available. Computers in Industry, Vol. 25, 1995, No. 3, pp. 309–330, doi: 10.1016/0166-3615(94)00044-Q.

[18] Pazienza, A.—Baselli, G.—Vinci, D. C.—Trussoni, M. V.: A Holistic Approach to Environmentally Sustainable Computing. Innovations in Systems and Software Engineering, Vol. 20, 2024, No. 3, pp. 347–371, doi: 10.1007/s11334-023-00548-9.

[19] Chatty, T.—Harrison, W.—Ba-Sabaa, H. H.—Faludi, J.—Murnane, E. L.: Co-Creating a Framework to Integrate Sustainable Design into Product Development Practice: Case Study at an Engineering Consultancy Firm. Sustainability, Vol. 14, 2022, No. 15, Art. No. 9740, doi: 10.3390/su14159740.

[20] Adigun, M. O.—Biyela, D. P.: Modelling an Enterprise for Re-Engineering: A Case Study. Proceedings of the 2003 Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists on Enablement Through Technology, 2003, pp. 153–164, doi: 10.1007/s11334-023-0058-9.

[21] Zada, I.—Shahzad, S.—Ali, S.—Mehmood, R. M.: OntoSuSD: Software Engineering Approaches Integration Ontology for Sustainable Software Development. Software: Practice and Experience, Vol. 53, 2023, No. 2, pp. 283–317, doi: 10.1002/spe.3149.

[22] Lan, Y. C.: Reengineering a Green Business. International Journal of Green Computing (IJGC), Vol. 2, 2011, No. 1, pp. 1–11, doi: 10.4018/jgc.2011010101.

[23] Ryan, C.: Automatic Re-Engineering of Software Using Genetic Programming. Springer New York, NY, 2000, doi: 10.1007/978-1-4615-4631-3.

[24] Rosenberg, L. H.—Hyatt, L. E.: Software Re-Engineering. Software Assurance Technology Center, Vol. 1, 1996, pp. 2–3.

[25] Tahvildari, L.—Kontogiannis, K.—Mylopoulos, J.: Quality-Driven Software Re-Engineering. Journal of Systems and Software, Vol. 66, 2004, No. 3, pp. 225–239, doi: 10.1016/S0164-1212(02)00082-1.

[26] Aggarwal, H. A.: Software Process Re-Engineering in SME's. Digitising Enterprise in an Information Age, CRC Press, 2021, pp. 185–196.

[27] Lopez-Herrejon, R. E.—Martinez, J.—Assunção, W. K. G.—Ziadi, T.—Acher, M.—Vergilio, S.: Handbook of Re-Engineering Software Intensive Systems into Software Product Lines. Springer, 2023, doi: 10.1007/978-3-031-11686-5.

[28] Penzenstadler, B.—Bauer, V.—Calero, C.—Franch, X.: Sustainability in Software Engineering: A Systematic Literature Review. 16th International Conference on Evaluation & Assessment in Software Engineering (EASE 2012), IET, 2012, pp. 32–41, doi: 10.1049/ic.2012.0004.

[29] Wolfram, N.—Lago, P.—Osborne, F.: Sustainability in Software Engineering. 2017 Sustainable Internet and ICT for Sustainability (SustainIT), IEEE, 2017, pp. 1–7, doi: 10.23919/SustainIT.2017.8379798.

[30] Danushi, O.—Forti, S.—Soldani, J.: Carbon-Efficient Software Design and Development: A Systematic Literature Review. ACM Computing Surveys, Vol. 57, 2025, No. 10, Art. No. 267, doi: 10.1145/3728638.

[31] Mahalakshmi, K.—Manikandan, S.—Nithyanantham, S.: Enhanced Software Engineering Process Model for Green Environment in View of Achieving Sustainabil-

ity. International Journal of Advanced Engineering Technology, Vol. 7, 2016, No. 2, pp. 1072–1079.

[32] Verdecchia, R.—Lago, P.—Ebert, C.—de Vries, C.: Green IT and Green Software. IEEE Software, Vol. 38, 2021, No. 6, pp. 7–15, doi: 10.1109/MS.2021.3102254.

[33] Karunaratne, S.—Dharmarathna, D.: A Review of Comprehensiveness, User-Friendliness, and Contribution for Sustainable Design of Whole Building Environmental Life Cycle Assessment Software Tools. Building and Environment, Vol. 212, 2022, Art. No. 108784, doi: 10.1016/j.buildenv.2022.108784.

[34] Hauschild, M. Z.—Kara, S.—Røpke, I.: Absolute Sustainability: Challenges to Life Cycle Engineering. CIRP Annals, Vol. 69, 2020, No. 2, pp. 533–553, doi: 10.1016/j.cirp.2020.05.004.

[35] Konersmann, M.—Borchers, J.—Bonorden, L.—Koch, A.—Schulze, S.: Towards a Software Reengineering Body of Knowledge (SREBOK). 2022, pp. 47–48.

[36] Khodabandehloo, H.—Roy, B.—Mondal, M.—Roy, C.—Schneider, K.: A Testing Approach While Re-Engineering Legacy Systems: An Industrial Case Study. 2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER), 2021, pp. 600–604, doi: 10.1109/SANER50967.2021.00073.

[37] Åstrand, A.: Re-Engineering a Database Driven Software Tool: Rebuilding, Automating Processes and Data Migration. Master Thesis. University of Vaasa, 2020.

[38] Fasching, C. M.: Towards a Framework for Risk Identification and Mitigation in Agile Software Re-Engineering: A Case Study. Ph.D. Thesis. University of Central Lancashire, 2023, doi: 10.17030/uclan.thesis.00052511.

[39] Lokhande, A.—Venkateswaran, C.—Ramachandran, M.—Chinnasami, S.—Vennila, T.: A Review on Various Implications on Re Engineering in Manufacturing. REST Journal on Emerging Trends in Modelling and Manufacturing, Vol. 7, 2021, No. 3, pp. 70–75, doi: 10.46632/7/3/1.

[40] Djan, E.—de Vries, M.: Business Process Re-Engineering and Agile Software Development: Applying the Story-Card Method. In: Hattingh, M., Matthee, M., Smuts, H., Pappas, I., Dwivedi, Y. K., Mäntymäki, M. (Eds.): Responsible Design, Implementation and Use of Information and Communication Technology (I3E 2020). Springer, Cham, Lecture Notes in Computer Science, Vol. 12066, 2020, pp. 370–382, doi: 10.1007/978-3-030-44999-5_31.

[41] Assunção, W. K. G.—Krüger, J.—Mendonça, W. D. F.: Variability Management Meets Microservices: Six Challenges of Re-Engineering Microservice-Based Webshops. Proceedings of the 24th ACM Conference on Systems and Software Product Line: Volume A (SPLC '20), 2020, doi: 10.1145/3382025.3414942.

[42] Brandtberg, R.: Re-Engineering Strategies for Legacy Software Systems. Master Thesis. University of Helsinky, 2020.

[43] Weerakkody, V.—Janssen, M.—El-Haddadeh, R.: The Resurgence of Business Process Re-Engineering in Public Sector Transformation Efforts: Exploring the Systemic Challenges and Unintended Consequences. Information Systems and e-Business Management, Vol. 19, 2021, No. 3, pp. 993–1014, doi: 10.1007/s10257-021-00527-2.

[44] Jiang, W.—Banna, V.—Vivek, N.—Goel, A.—Synovic, N.—Thiruvathukal, G. K.—Davis, J. C.: Challenges and Practices of Deep Learning Model Reengineering: A Case Study on Computer Vision. Empirical Software Engineering, Vol. 29, 2024, No. 6, Art. No. 142, doi: 10.1007/s10664-024-10521-0.

[45] Ahmed, S. I.—Ahmed, A. A.—Yaseen, M. H.: Analysis the Business Process Re-Engineering to Support Sustainability. POODA Journal of Business Marketing, Finance, and Accounting Studies, Vol. 14, 2024, No. 2, https://pooda.org/art/22024i.pdf.

[46] Atadoga, A.—Umoga, U. J.—Lottu, O. A.—Sodiya, E. O.: Tools, Techniques, and Trends in Sustainable Software Engineering: A Critical Review of Current Practices and Future Directions. World Journal of Advanced Engineering Technology and Sciences, Vol. 11, 2024, No. 1, pp. 231–239, doi: 10.30574/wjaets.2024.11.1.0051.

[47] Nadeem, M. A.—Lee, S. U. J.—Younus, M. U.: A Comparison of Recent Requirements Gathering and Management Tools in Requirements Engineering for IoT-Enabled Sustainable Cities. Sustainability, Vol. 14, 2022, No. 4, Art. No. 2427, doi: 10.3390/su14042427.

[48] Ponnusamy, S.—Eswararaj, D.: Navigating the Modernization of Legacy Applications and Data: Effective Strategies and Best Practices. Asian Journal of Research in Computer Science, Vol. 16, 2023, No. 4, pp. 239–256, doi: 10.9734/ajrcos/2023/v16i4386.

[49] Mishra, A. K.—Nagpal, R.—Seth, K.—Sehgal, R.: A Critical Review on Service Oriented Architecture and Its Maintainability. 2021 9th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO), 2021, pp. 1–8, doi: 10.1109/ICRITO51393.2021.9596285.

[50] Heričko, T.—Heričko, M.—Brdnik, S.: Software Sustainability Requirements for Knowledge Management Systems in the Cultural Heritage Domain. In: Uden, L., Ting, I. H. (Eds.): Knowledge Management in Organizations (KMO 2023). Springer, Cham, Communications in Computer and Information Science, Vol. 1825, 2023, pp. 302–313, doi: 10.1007/978-3-031-34045-1_25.

[51] Karita, L.—Mourão, B. C.—Machado, I.: Towards a Common Understanding of Sustainable Software Development. Proceedings of the XXXVI Brazilian Symposium on Software Engineering, 2022, pp. 269–278, doi: 10.1145/3555228.3555236.

[52] Kaur, K.—Kaur, J.—Singh, R.: Re-Engineering Education and Training: Fostering Digitalization for Sustainability. Digital Analytics Applications for Sustainable Training and Education, Apple Academic Press, 2024, pp. 191–209, doi: 10.1201/9781032713366.

[53] Bustamante-Mora, A.—Diéguez-Rebolledo, M.—Hormazábal, Y.—Millar, L.—Cadena, R.: Challenges and Opportunities for Sustainable Engineering: Products, Services, Technologies, and Social Inclusivity with a Gender Approach. Sustainability, Vol. 16, 2024, No. 5, Art. No. 1888, doi: 10.3390/su16051888.

[54] Nath, I.: Mapping Software Engineering Components (SE3) and Risk Management for Optimum Release Re-Engineering. ResearchGate, 2020, doi: 10.13140/RG.2.2.30601.36962.

[55] Al-Anqoudi, Y.—Al-Hamdani, A.—Al-Badawi, M.—Hedjam, R.: Using Ma-

chine Learning in Business Process Re-Engineering. Big Data and Cognitive Computing, Vol. 5, 2021, No. 4, Art. No. 61, doi: 10.3390/bdcc5040061.

[56] Laplante, P. A.—Kassab, M.: What Every Engineer Should Know About Software Engineering. 2nd Edition. CRC Press, 2022, doi: 10.1201/9781003218647.

[57] Semerikov, S. O.—Striuk, A. M.—Striuk, L.—Striuk, M.—Shalatska, H.: Sustainability in Software Engineering Education: A Case of General Professional Competencies. The International Conference on Sustainable Futures: Environmental, Technological, Social and Economic Matters (ICSF 2020), E3S Web of Conferences, 2020, doi: 10.1051/e3sconf/202016610036.

[58] Nazir, S.—Fatima, N.—Chuprat, S.: Situational Factors for Modern Code Review to Support Software Engineers' Sustainability. International Journal of Advanced Computer Science and Applications (IJACSA), Vol. 11, 2020, No. 1, pp. 498–504, doi: 10.14569/IJACSA.2020.0110161.

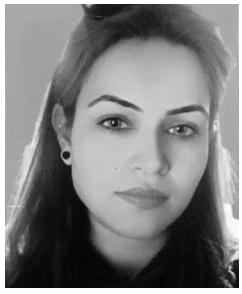
Islam Zada is Assistant Professor in the Department of Software Engineering at the International Islamic University Islamabad (IIUI), Pakistan. He received his Ph.D. degree in computer science, with a specialization in software engineering, from the University of Peshawar, Pakistan. He has extensive experience in teaching, research, and supervision at undergraduate and postgraduate levels. His research interests include sustainable software engineering, green computing, Internet of Things (IoT), and smart cities. He has published numerous research articles in reputable international journals and conferences. He also serves as an editor and reviewer for more than 20 international journals, contributing actively to the academic and research community.

Hessa Alfraihi is Assistant Professor in the Information Systems Department at the University of Princess Nourah bint Abdulrahman University, Saudi Arabia. She has completed her Ph.D. in software engineering at the Kings College London in 2020. At present, her research endeavors are centered around cutting-edge areas of software engineering, with a primary focus on model-driven software engineering, agile development, and machine learning. Her work in these domains demonstrates her visionary approach to tackling complex software development challenges through innovative methods and tools.

Sara Shahzad received her Ph.D. degree in computer science, with the specialization in agile software development. She is currently Professor with the Department of Computer Science, University of Peshawar. She has almost 20 years of experience in teaching at graduate and postgraduate levels, and leads the Software Engineering Research Group, Department of Computer Science. Her group is actively working in different areas such as software quality, program comprehension and software complexity, software cloning and theft detection, data analytics, and agile software development. She has a special interest in using agile methods for software engineering education and for the professional grooming of students.

Abdullah Alsaeedi is Associate Professor in the Department of Computer Science at the Taibah University, Madinah, Saudi Arabia. He received his Ph.D. degree in computer science, from the University of Sheffield, UK, in 2016. His research interests include software engineering, software model inference, grammar inference, machine learning, NLP, sentiment analysis, medical imaging and AI application. He has published numerous research articles in reputable international journals and conferences.

Manal Aldhayan is affiliated with the Department of Computer Science, College of Computer and Information Technology, Shaqra University, Shaqra, Saudi Arabia. She has extensive experience in teaching and research in the field of computer science. Her research interests include artificial intelligence, data science, software engineering, and emerging computing technologies. She has contributed to several research publications in reputable international journals and conferences and is actively involved in academic and research activities.



Attiya Kanwal is Assistant Professor in the Department of Bioinformatics at the International Islamic University's Faculty of Computing and Information Technology, holds her Ph.D. in bioinformatics from the Capital University of Science and Technology. She specializes in computer-aided drug design, meta-analysis, gene expression profiling, protein-protein interaction network analysis, and systems biology, utilizing her proficiency in several programming languages to analyze complex biological data. With a robust commitment to advancing bioinformatics, she engages in pioneering research addressing key biomedical challenges and contributes to the development of innovative computational tools. Passionate about both teaching and research, she inspires and mentors future bioinformatics professionals, fostering a stimulating learning environment. Beyond her academic endeavors, she continuously seeks to expand her expertise, actively participating in conferences, workshops, and collaborative initiatives to stay at the forefront of advancements in the field.

Safia Zaheer is currently pursuing her Master's degree in data science at the Air University, Islamabad, Pakistan. She holds her Bachelor's degree in bioinformatics from the International Islamic University, Islamabad, Pakistan. Her undergraduate research project focused on the identification of potential inhibitors for alkaptonuria using molecular dynamics simulations. During her Master's studies, she has worked on projects including deepfake detection, sentiment analysis, and fake news detection and malicious user detection. Her research interests include artificial graph based neural networks, data visualization machine learning, deep learning, natural language processing, and data science applications.

Hussein A. Al-Hashimi is Assistant Professor and Head of the Educational Affairs Unit at CCIS King Saud University, specializing in software engineering. He earned his Ph.D. from the University of Southampton. With a rich blend of academic and practical experience, he actively contributes as a part-time consultant in cyber security, AI, and digital transformation for both the government and private sectors. His expertise and insights have significantly impacted the academic community and industry practices alike, establishing him as a thought leader in his areas of specialization.