الجمهورية الجزائرية الديموقراطية الشعبية
People's Democratic Republic of Algeria

وزارة التعليم العالي والبحث العلمي
Ministry of Higher Education and Scientific Research

المدرسة الوطنية العليا للتكنولوجيات المتقدمة
National Higher School of Advanced Technologies

Department of Industrial Engineering

Final Year Project to Obtain the Diploma of Engineering

-Field-

Industrial Engineering

-Specialty-

Industrial Engineering

- Subject-

Optimization of a Software Development Process for Sustainability

Case study: Icosnet SPA

Realized by

YOUSFI Abderrahmane

Presented publicly, the 24 / 06 / 2025

Members of The Jury :

First LAST NAME	University	Grade	Quality
Mr. LAIDI Mohamed	ENSTA	MCB	President
Mrs. GHOMARI Leila	ENSTA	MCA	Supervisor
Mrs. BELAYADI Djahida	ENSTA	MCB	Co-supervisor
Mr. REZGUI Wail	ENSTA	MCB	Examinator

2024/2025

Dedication

To my parents, thank you for your unwavering love and support. Your sacrifices, patience, and belief in me made this possible. I carry your strength and values with me every step of the way.

To my family and friends, thank you for being my constant through the highs and lows. Your encouragement, humor, and presence have meant more than I can put into words.

To my professors and advisors, your guidance and generosity shaped not just my work, but the way I think and grow. I am grateful for every lesson, challenge, and conversation.

This project is not just mine, it is built on everything you have given me. Thank you.

Thanks

I would like to express my deepest gratitude to my advisor, Professor Ghomari, for her unwavering support and invaluable guidance throughout this project. Her expertise, dedication, and belief in my work played a crucial role in bringing it to life.

I am also sincerely thankful to my co-advisor, Professor Belayadi, whose insights and encouragement helped shape the direction of my research and pushed me to think critically at every step.

A special thanks to Aymen, my internship advisor, for his support and thoughtful advice during such a pivotal phase of my academic journey. I'm also grateful to Icosnet and the entire development team for providing a collaborative and enriching environment that greatly enhanced my learning experience.

To all the professors who have taught and guided me over the years—thank you. Your dedication and passion for teaching have left a lasting impact on both my academic and personal growth.

And finally, to my parents—your constant love, patience, and belief in me have been the foundation of everything I've accomplished. This achievement belongs to you as much as it does to me.

Thank you all for being part of this journey.

الملخص

في ظل التحديات البيئية والاقتصادية المتزايدة، أصبح تحقيق التنمية المستدامة أولوية رئيسية في مجالات الهندسة وأنظمة المعلومات. ومن هذا المنطلق، تظهر هندسة البرمجيات المستدامة كفرع يسعى إلى تصميم برمجيات فعالة ومسؤولة ومتوافقة مع أهداف الاستدامة طويلة المدى. يتناول هذا البحث كيفية دمج مبادئ هندسة البرمجيات المستدامة مع منهجيات هندسية لتحسين أداء واستدامة دورة حياة تطوير البرمجيات، وهو إطار عمل منظم يوجه مراحل تخطيط وتنفيذ واختبار وصيانة الأنظمة البرمجية. يستند البحث إلى دراسة حالة لمشروع تطوير منصة تجارة إلكترونية في الجزائر. بعد تحليل أسباب ضعف عملية التطوير الحالية، مثل التصلب وعدم إشراك العميل وغياب معايير الاستدامة، تم إجراء مقارنة متعددة المعايير لاختيار المنهجية الأنسب، حيث تم اختيار منهجية سكرم نظراً لمرونتها ونهجها التعاوني. أُجريت محاكاة لتوقع و تقييم تأثير هذا الدمج على مؤشرات الأداء الرئيسية مثل سرعة الفريق، ساعات إعادة العمل، الدين التقني، ورضا العملاء. وقد أظهرت النتائج أن اعتماد سكرم مع مبادئ هندسة البرمجيات المستدامة يساهم في تحسين جودة البرمجيات واستجابتها واستدامتها.

الكلمات المفتاحية: هندسة البرمجيات المستدامة، دورة حياة تطوير البرمجيات، سكرم، عملية تطوير البرمجيات

Abstract

Faced with growing environmental and economic challenges, sustainable development has become a central concern in engineering and information systems. In this context, Sustainable Software Engineering (SSE) emerges as a discipline aiming to design software that is efficient, responsible, and aligned with long-term sustainability goals. This thesis investigates how the integration of SSE principles with engineering methodologies can improve the performance and sustainability of the Software Development Lifecycle (SDLC), a structured framework that guides the planning, implementation, testing, and maintenance of software systems. The study is based on a case project involving the development of an e-commerce platform in Algeria. After analyzing the limitations of the company's software development processes, including its rigidity and lack of client interaction and sustainability measures, several methods and tools were evaluated using a multi-criteria comparison. Scrum was selected for its adaptability and collaborative structure. A simulation was conducted to predict and assess the impact of this integration on key project indicators such as team velocity, rework hours, technical debt, and customer satisfaction. The results demonstrate that applying Scrum with SSE principles improves responsiveness, quality, and sustainability in software projects

Keywords: Sustainable Software Engineering, SDLC, Scrum, Software Development Process

Résumé

Face aux défis environnementaux et économiques croissants, le développement durable est devenu une priorité centrale dans les domaines de l'ingénierie et des systèmes d'information. Dans ce contexte, l'ingénierie logicielle durable émerge comme une discipline visant à concevoir des logiciels efficaces, responsables et alignés sur des objectifs de durabilité à long terme. Ce mémoire étudie comment l'intégration des principes de la SSE avec des méthodologies d'ingénierie permet d'améliorer la performance et la durabilité du cycle de vie du développement logiciel, un cadre structuré qui guide la planification, l'implémentation, les tests et la maintenance des systèmes logiciels. L'étude est fondée sur un projet de développement d'une plateforme e-commerce en Algérie. Après une analyse des limites du processus de développement de l'entreprise, notamment sa rigidité, le manque d'interaction avec le client et l'absence de prise en compte de la durabilité, plusieurs méthodes ont été comparées selon une approche multicritère. La méthode Scrum a été choisie pour sa flexibilité et son approche collaborative. Une simulation a été réalisée pour prédire et évaluer l'impact de cette intégration sur des indicateurs clés tels que la vélocité de l'équipe, les heures de reprise, la dette technique et la satisfaction client. Les résultats montrent que l'application de Scrum avec les principes de la SSE permet d'améliorer la réactivité, la qualité et la durabilité des projets logiciels.

Mots-clés : Ingénierie Logicielle Durable, cycle de vie du développement logiciel, Scrum, Processus de Développement Logiciel

Contents

List of Abbreviations **3**

General Introduction **4**

1 Foundations of Sustainable Software Engineering **6**
1.1 CO2 Emissions and IT . . . 6
1.1.1 CO2 Consumption Caused by IT . . . 6
1.1.2 CO2 Savings from IT . . . 7
1.2 The Software Development Lifecycle . . . 7
1.2.1 The Waterfall Model . . . 8
1.2.2 Agile Development Approaches . . . 9
1.2.3 Comparison : Waterfall vs. Agile . . . 9
1.3 Sustainability in Software Engineering . . . 10
1.4 Sustainable Techniques and Tools Mapped to SDLC Phases . . . 11
1.4.1 Value Stream Mapping . . . 11
1.4.2 Kanban . . . 12
1.4.3 Scrum . . . 12
1.4.4 Life Cycle Assessment . . . 13
1.5 Conclusion . . . 15

2 Case Study : E-Commerce Platform Project **16**
2.1 Presentation of the Internship Hosting Company . . . 16
2.2 Development Team and Department Structure . . . 17
2.3 Project overview . . . 19
2.4 Current SDLC Model . . . 19
2.4.1 Current Process – Requirement Analysis Phase . . . 20
2.4.2 Current Process – Design Phase . . . 20
2.4.3 Current Process – Implementation Phase . . . 21
2.4.4 Current Process – Testing Phase . . . 22
2.4.5 Current Process – Deployment Phase . . . 22
2.4.6 Current Process – Maintenance Phase . . . 22
2.5 Diagnostics of the Current Software Development Process . . . 23
2.5.1 SWOT Analysis . . . 24
2.5.2 Ishikawa Diagram . . . 24
2.5.3 The 5 Whys Method . . . 25
2.5.4 Diagnostic Results and Improvement Proposals . . . 26
2.6 Methodological Selection Approach . . . 30
2.6.1 Criteria for Method Selection . . . 30
2.6.2 Multicriteria Decision Matrix . . . 32
2.6.3 Final Selection Based on Decision Matrix . . . 32

2.7 Conclusion 32

3 Scrum Implementation in the SDLC **34**
3.1 Scrum Framework Overview 34
3.2 The Proposed SDLC 35
3.3 Project Simulation with Scrum 35
3.3.1 Step 1 : Setup and Team Alignment 35
3.3.2 Step 2 : Sprint Execution and Feature Prioritization 36
3.3.3 Step 3 : Continuous Improvement and Sustainability Integration . . 40
3.4 Conclusion 41

4 Evaluation and Predictive Impact of Sustainable Scrum Adoption **42**
4.1 Dataset Overview 42
4.1.1 Data Creation 42
4.1.2 Dataset Description 43
4.2 Prediction Objectives 44
4.3 Data Processing 44
4.4 Model Selection and Evaluation 45
4.5 Prediction Results Analysis 47
4.6 Limitations 48
4.7 Conclusion 49

General Conclusion **50**

Appendices **52**
Appendix A : Business Model Canvas 53
Appendix B : Scrum Framework 54
Appendix C : Roles organization in Scrum 56
Appendix D : Data sample 57
Appendix E : Planning Poker Estimation Technique 59

Bibliography **59**

List of Figures

1.1 Power Usage Effectiveness . . . 8
1.2 A Typical SDLC . . . 9
1.3 Mapping Engineering Methods to the SDLC Phases . . . 11
1.4 Kanban Board . . . 12
1.5 Scrum Process . . . 13
1.6 Use of SimaPro in LCA . . . 13

2.1 ICOSNET's Solutions (internal document) . . . 17
2.2 ICOSNET's Organizational Hierarchy . . . 18
2.3 High Level SDLC Model . . . 20
2.4 Requirements Analysis Phase . . . 20
2.5 Design Phase Process . . . 21
2.6 Implementation Phase Process . . . 21
2.7 Testing Phase Process . . . 22
2.8 Deployment Phase Process . . . 23
2.9 SWOT matrix applied to the current development process. . . . 27
2.10 the Ishikawa Diagram for the Current Development Process . . . 28
2.11 The Two 5 Whys Diagrams Used in Diagnostic Analysis . . . 29

3.1 Transition to Scrum . . . 34
3.2 Scrum Implementation Plan . . . 35
3.3 The Product Backlog . . . 37
3.4 Scrum Board . . . 39
3.5 The Sprint Tracking Sheet . . . 39
3.6 Sprint Review Process According to the SBOK . . . 40
3.7 The Burndown Chart for Sprint 1 . . . 40

4.1 Automatic data preprocessing in Orange . . . 44
4.2 Comparison of Regression Models . . . 45
4.3 Models Testing Workflow Using Orange DM . . . 46
4.4 AdaBoost Workflow . . . 47
4.5 Improvement of SSE Metrics After Scrum & Sustainability Integration . . . 48
B.6 Scrum Cheat Sheet . . . 55
C.7 Scrum Roles . . . 56
D.8 Dataset Sample . . . 58

List of Tables

1.1 Key Differences Between Waterfall and Agile Development Models 10
1.2 Aligning sustainability methods and their impact with SDLC phases. . . . 14

2.1 E-Commerce Project Overview 19
2.2 Observed Issues, Root Causes, and Improvement Proposals 26
2.3 Multicriteria Decision Matrix 32

3.1 Team Roles and Responsibilities 36
3.2 MoSCoW Prioritization of Project Requirements 36
3.3 Sprint 1 – User Stories 38

4.1 Project Parameters Description 43
4.2 Prediction Metrics and Sustainability Dimensions 44
4.3 Key Performance Metrics: Before and After Scrum + SSE 47
1.4 Business Model Canvas 53

List of Abbreviations

AI	Artificial Intelligence
BPMN	Business Process Model and Notation
CI/CD	Continuous Integration / Continuous Deployment
CO2	Carbon Dioxide
ICT	Information and Communication Technology
IT	Information Technology
ISP	Internet Service Provider
KPI	Key Performance Indicator
LCA	Life Cycle Assessment
MAE	Mean Absolute Error
ML	Machine Learning
MoSCoW	Must, Should, Could, Won't (priority method)
MSE	Mean Squared Error
PMO	Project Management Office
PO	Product Owner
QA	Quality Assurance
R^2	R-Squared (Coefficient of Determination)
RMSE	Root Mean Squared Error
SBOK	Scrum Body Of Knowledge
SM	Scrum Master
SME	Small and Medium Enterprise
SDLC	Software Development Life Cycle
SaaS	Software as a Service
SSE	Sustainable Software Engineering
SWOT	Strengths, Weaknesses, Opportunities, Threats
SVM	Support Vector Machine
UAT	User Acceptance Testing
UI	User Interface
UX	User Experience
WIP	Work In Progress
XP	Extreme Programming

General Introduction

In today's increasingly digital world, software systems play a central role in shaping business operations, user experience, and environmental impact. As digital platforms continue to grow in complexity and scale, organizations must balance technical performance with sustainable development goals. This need has given rise to Sustainable Software Engineering (SSE) an approach that integrates environmental, economic, and social responsibility into the entire software lifecycle [35, 45]. When combined with agile practices such as Scrum, SSE enables more adaptable, energy-efficient, and user-centered software systems [39].

This project was conducted at ICOSNET, a leading Algerian IT service provider. The core objective was to design and simulate a sustainable development workflow for an e-commerce platform tailored to the Algerian market. The project specifically targeted the limitations of ICOSNET's current development process, which follows a development process model characterized by low client involvement, rigid planning, and lack of sustainability integration. Through our simulation, we explored how Scrum, supported by SSE principles, could address these limitations and promote continuous improvement, early feedback, and resource optimization [28].

Throughout the project, various tools and diagnostic methods were applied to evaluate both technical and sustainability performance. The methodology combined theoretical foundations with practical implementation. Techniques such as SWOT analysis, Ishikawa diagram, and the 5 Whys method were used to diagnose inefficiencies. Later, a set of sustainability-oriented methods, including Scrum, Kanban, Value Stream Mapping (VSM), and Life Cycle Assessment (LCA), were evaluated using a multi-criteria decision matrix [24, 11]. Based on the results, Scrum was adopted as the core framework for the simulation objectives [2].

The practical implementation of Scrum was carried out across the most resource-intensive phases of the Software Development Lifecycle (SDLC): design, implementation, testing, and deployment [12]. Each sprint was carefully planned and reviewed using real project data, prioritization techniques such as MoSCoW, and burndown charts [42].

This research contributes to a growing body of work that demonstrates the feasibility of integrating sustainability into software development models [46]. By combining SSE principles with Scrum, the project showed how even small teams working within constrained environments can improve delivery rhythm, reduce waste, and maintain user-centric development [37].

This thesis is structured as follows:

- **Chapter 1:** Foundations of Sustainable Software Engineering It begins by assessing the environmental impact of IT systems, then introduces core concepts in SSE, explains its importance across the SDLC, and presents a selection of sustainable tools such as Scrum, VSM, Kanban, and LCA.
- **Chapter 2:** Case Study – E-Commerce Platform Project Describes the business context, outlines project needs and challenges, and analyzes the current development model using SSE diagnostic tools and method selection criteria.
- **Chapter 3:** Optimized SDLC with Scrum Details the practical simulation of Scrum, including sprint setup, backlog management, and feedback cycles, showing how the team adapted agile principles to improve process flow.
- **Chapter 4:** Evaluation and Predictive Impact of Sustainable Scrum Adoption Discusses the predicted and expected outcomes of using Scrum and SSE practices, such as rework hours, team velocity, and client satisfaction, based on observed metrics and sprint performance.

Chapter 1

Foundations of Sustainable Software Engineering

As digital technologies continue to shape modern life, the reliance on software is increasing, and with it, the environmental, economic, and social impacts of software systems. While IT can play a central role in advancing sustainability, achieving real progress requires that software itself be developed with sustainability principles in mind. SSE is an emerging discipline that focuses on designing and managing software systems in alignment with long-term sustainability goals [46].

This chapter introduces the foundations of SSE. It begins by presenting the environmental footprint of IT, including the CO2 emissions it generates and the potential for CO2 savings through digital solutions. We also define the Software Development Lifecycle (SDLC), which serves as the main framework through which sustainability can be applied in practice. Furthermore, the chapter includes a comparison between traditional Waterfall and Agile approaches to software development, highlighting their structural differences and implications for sustainable practices.The chapter also outlines how software engineering affects and can contribute to sustainability across four dimensions: environmental, economic, social, and technical. Together, these elements form the groundwork for integrating sustainability into software development in a structured and measurable way.

1.1 CO2 Emissions and IT

Digital technology becomes deeply integrated into everyday life, the environmental impact of the IT sector is growing rapidly. While it enables efficiency and productivity, it also requires significant energy and resources, which results in CO2 emissions. This section will examine the CO2 emissions associated with IT and the potential CO2 savings that can be achieved through IT. Where it serves as the motivation behind this project.

1.1.1 CO2 Consumption Caused by IT

Every digital activity, like browsing, streaming, or using the cloud, consumes energy and generates CO2, especially when powered by non-renewable sources.
Some examples :

- Data centres alone account for about 1% of global electricity use, potentially ris-

ing to 8% by 2030 without changes [32]. These facilities require massive energy, particularly for cooling.

- Networks that support internet and mobile services are also energy-intensive. Transferring just 1 GB of data can consume 5 kWh and emit 1.5 kg of CO2 if powered by coal [7]. Without intervention, the ICT sector could contribute over 14% of global CO2 emissions by 2040 [9].
- Devices like desktops and smartphones add to this impact. Producing a desktop emits 120–250 kg of CO2, with yearly use adding up to 175 kg. Their short lifespan increases this footprint [9].

1.1.2 CO2 Savings from IT

While IT contributes to CO2 emissions, it also has significant potential to reduce emissions across other sectors. IT can enhance industrial operations, transportation, and energy management efficiency, promising a more sustainable future [20].
Some studies include:

- IT has the power to optimise power grids [29], dynamically matching supply and demand to reduce energy waste. It has been demonstrated to reduce energy consumption by up to 10% in certain regions [23].
- IT solutions in transportation, such as intelligent traffic systems and fleet optimisation, can reduce fuel consumption. These systems have the potential to reduce CO2 emissions in urban areas in the future. Ride-sharing applications, autonomous vehicles, and telecommuting have also reduced emissions [20].
- Smart Manufacturing: Iot sensors and predictive maintenance allow factories to operate more efficiently. Manufacturers can reduce energy use and prevent costly equipment breakdowns by monitoring machinery and optimising processes. This approach, often referred to as lean manufacturing, has the potential to reduce operational CO2 emissions [20, 29]

As Figure 1.1 illustrates, IT contributes significantly to CO2 emissions through data centres, networks, and device manufacturing. However, IT also offers solutions for CO2 savings, particularly in the power management, transportation, and industrial sectors. By implementing these strategies, IT can help reduce its environmental impact and play a crucial role in achieving global sustainability goals.

1.2 The Software Development Lifecycle

The Software Development Lifecycle (SDLC) is a structured process that guides the development of software from initial concept to deployment and maintenance. It helps teams organize work, allocate resources, and track project progress efficiently [27, 47]. Choosing the right SDLC model significantly impacts the success and sustainability of a project. A standard SDLC consists of six key phases[12, 27, 16], each with a specific purpose, as shown in Figure 1.2:

1. **Requirement Gathering and Analysis:** Stakeholders define the project's goals and capture both functional and non-functional requirements. This step builds a solid foundation for the rest of the process.
2. **Design:** Based on the analysis, the team designs system architecture, user interfaces, and technical components. Prototypes may be developed to validate early

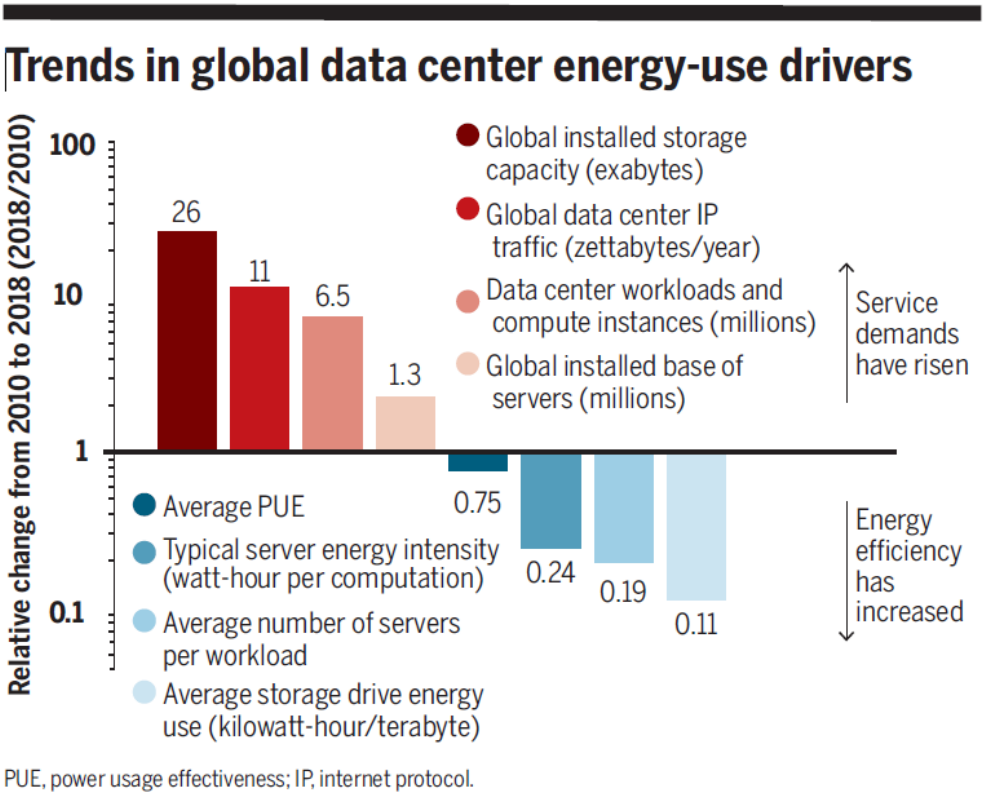


Figure 1.1: Power Usage Effectiveness

concepts with stakeholders.

3. **Implementation or Coding:** Developers write the actual code, following the design plan. Unit testing is performed to ensure components function as expected.
4. **Testing:** The software is tested through multiple levels (e.g., system, integration, UAT) to identify bugs and verify that requirements are met before release.
5. **Deployment:** Once validated, the software is deployed to the production environment. Final adjustments may be made based on live testing.
6. **Maintenance:** Post-deployment, the team monitors performance and addresses user issues, applying updates to keep the software secure and functional.

Understanding the SDLC is essential for integrating sustainability, as each phase presents unique opportunities to reduce waste, optimize resources, and improve long-term maintainability. The next section explores how Sustainable Software Engineering aligns with and improves this lifecycle.

1.2.1 The Waterfall Model

The Waterfall model is one of the earliest and most widely used software development models, characterized by a linear and sequential flow of phases [3]. Each stage must be completed before the next one begins, with minimal overlap or iteration. A typical Waterfall process follows the steps of requirement analysis, design, implementation, testing, deployment, and maintenance, which aligns with the standard SDLC structure described earlier.

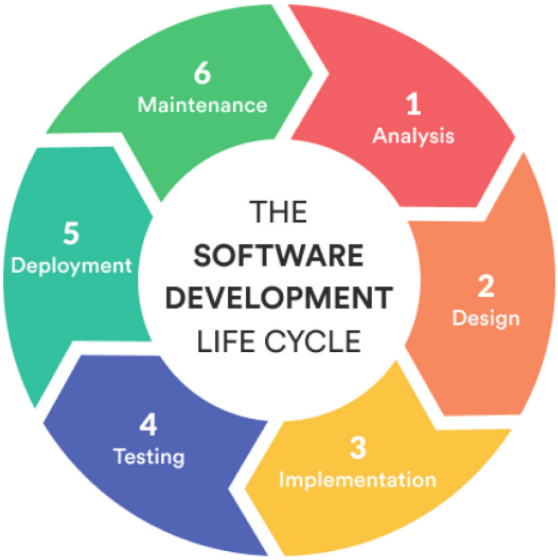


Figure 1.2: A Typical SDLC

This model provides a clear and disciplined framework for software development, particularly suited for projects with well-defined requirements and low expected volatility. It ensures that each deliverable is reviewed and approved before moving forward, offering predictability, documentation, and control over progress. These advantages have made it popular in enterprise environments and public sector projects [27, 36].

1.2.2 Agile Development Approaches

While traditional models like Waterfall follow a linear and phase-driven structure, Agile represents a more flexible and adaptive mindset toward software development. Agile is not a single method, but rather a collection of iterative, incremental approaches that emphasize collaboration, responsiveness, and continuous value delivery[36].
Agile frameworks, including Scrum, and XP, share key characteristics: short development cycles, evolving requirements, cross-functional teams, and direct user involvement. Instead of fully defining requirements upfront, Agile embraces change and encourages frequent reassessment of priorities based on feedback and working software [27].
At the heart of Agile is the idea of delivering value early and regularly. Each iteration includes planning, coding, testing, and demonstration of potentially shippable features [39]. This cycle promotes rapid learning, higher client satisfaction, and better alignment with real-world needs.
In Agile environments, documentation is streamlined, and team communication is prioritized over formal processes. Governance exists, but it is lightweight, where it focuses more on outcomes than procedures. This enables teams to adapt quickly to changes without sacrificing quality [36].

1.2.3 Comparison: Waterfall vs. Agile

To highlight the contrast between traditional and modern development approaches, Table 1.1 compares the Waterfall and Agile models across key dimensions relevant to software engineering and sustainability [36, 17].

Table 1.1: Key Differences Between Waterfall and Agile Development Models

Aspect	**Waterfall Model**	**Agile Development**
Process Structure	Sequential, phase-based	Iterative and incremental
Requirement Handling	Fully defined at the beginning	Evolving throughout the project
User Involvement	Late (mostly at testing/UAT phase)	Continuous (in every iteration or sprint)
Flexibility to Change	Low – changes are costly once development starts	High – frequent re-evaluation and reprioritization
Delivery Style	One-time delivery at the end	Frequent delivery of working software
Documentation	Comprehensive and predefined	Minimal, just enough for clarity
Team Structure	Predefined, often hierarchical	self-organizing

This comparison illustrates the fundamental differences between two dominant software development paradigms. The Waterfall model provides clarity and control but lacks the adaptability required for fast-changing environments like e-commerce projects. In contrast, Agile approaches prioritize responsiveness, user involvement, and continuous delivery, those characteristics align closely with the goals of Sustainable Software Engineering[17].

1.3 Sustainability in Software Engineering

Sustainability in software engineering involves designing, building, and maintaining systems that meet present needs without compromising future viability. It aims to minimize negative impacts across environmental, economic, social, and technical dimensions [46, 35, 25].
Key aspects of Sustainable Software Engineering (SSE) include:

- **Environmental Impact:** Reducing energy consumption and hardware demands helps reduce CO2 emissions while maintaining performance [32, 35].
- **Economic Efficiency:** Sustainable systems are designed to be maintainable and scalable, minimizing the cost of redesigns and ensuring long-term financial value [45].
- **Social Responsibility:** Inclusive and accessible software promotes equity and supports diverse user needs, aligning with ethical digital development [45].
- **Technical Longevity:** Systems built to be modular, well-documented, and adaptable reduce the risk of obsolescence and electronic waste, supporting long-term usability [45, 34].

By applying these principles, developers can build software that is not only functional but also capable of evolving over time with minimal environmental footprint and maximum value for users and businesses.

1.4 Sustainable Techniques and Tools Mapped to SDLC Phases

This section examines engineering techniques and their integration into the phases of the SDLC. The goal is to incorporate methodologies that improve efficiency, reduce waste, and minimize environmental impact at each stage of the SDLC. Four methods will be introduced: VSM, Scrum, Kanban, and LCA. The section will also explore the impact of these methods on SSE, mapping each method to the SDLC phases where it can have the most significant effect. Based on studies and use cases, the best application of each method will be discussed, alongside the metrics that can assess sustainability for each approach.
Figure 1.3 highlights how each methodology can be mapped to specific stages of the lifecycle to maximize sustainability for software development.

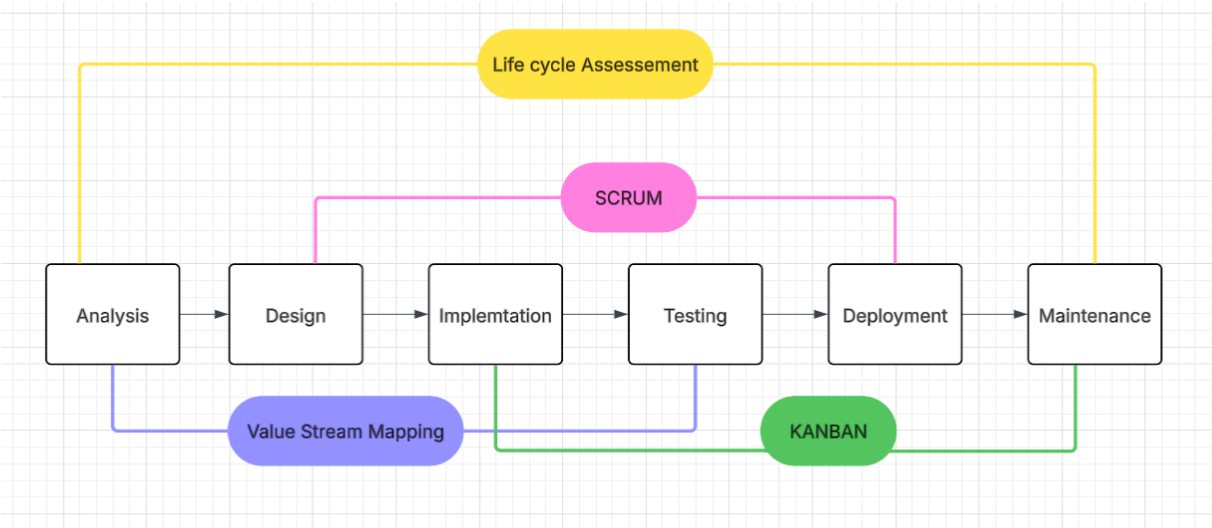


Figure 1.3: Mapping Engineering Methods to the SDLC Phases

1.4.1 Value Stream Mapping

Value Stream Mapping (VSM) is a key tool in Lean methodology that visualizes, analyzes, and optimizes the flow of materials and information within a process. In the context of software development, VSM is valuable for identifying both value-added and non-value-added activities, helping to streamline operations, reduce waste, and improve efficiency throughout the SDLC [24].
VSM starts by creating a "current state map" that outlines the existing flow of tasks, identifying bottlenecks, delays, and inefficiencies. By calculating the time spent on value-added versus non-value-added activities, teams can design a "future state map" that incorporates improvements such as automation, process simplification, and better communication [43].
In the context of sustainability, VSM plays a crucial role by pinpointing inefficiencies and optimizing resource usage. It helps reduce waste in terms of energy, time, and materials by eliminating non-value-added activities like unnecessary waiting times or redundant tasks. This method supports long-term sustainability by encouraging iterative improvements and reducing the need for extensive rework, which can have significant environmental and operational impacts.

1.4.2 Kanban

Kanban is a visual workflow management tool used in the SDLC to optimize processes and enhance sustainability. It helps teams visualize tasks, limit Work In Progress (WIP), and manage flow, ensuring efficient use of resources and reducing waste [39]. The Kanban board, a core component of this tool, provides a clear view of the development process, as shown in Figure 1.4, which illustrates how a Kanban board is applied in a software development lifecycle. This visualization enables teams to identify bottlenecks and prioritize tasks effectively [2].

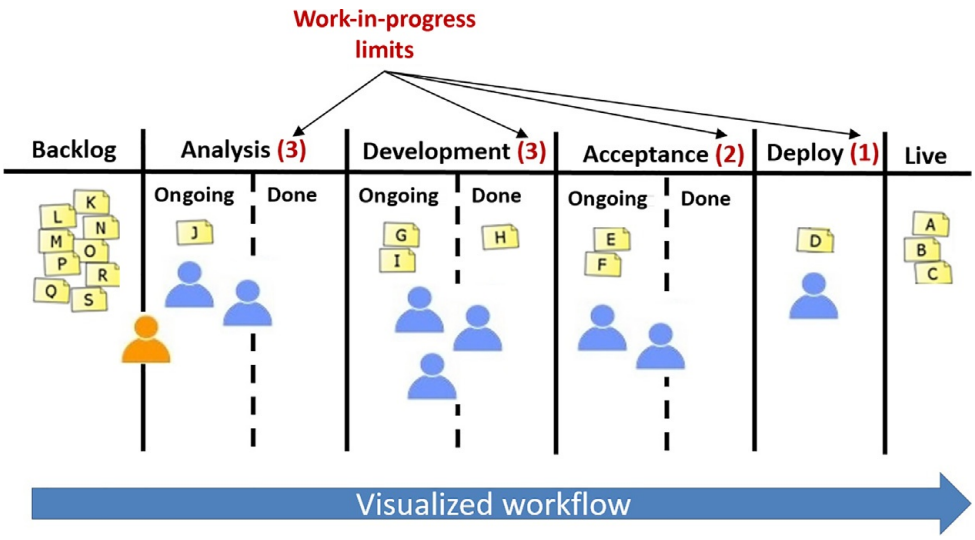


Figure 1.4: Kanban Board

By implementing WIP limits, Kanban prevents bottlenecks, improves workflow, and discourages teams from working on too many tasks simultaneously. This approach reduces stress, minimizes inefficiencies, and ensures a focused and sustainable development process. Additionally, the emphasis on completing tasks before starting new ones helps to minimize energy and resource consumption, aligning with sustainable practices.
Kanban also supports sustainability by promoting a lean approach, avoiding overproduction, reducing defects, and optimizing resource allocation. It ensures that the development process remains efficient, environmentally friendly, and focused on delivering value to the customer [8].

1.4.3 Scrum

Scrum is an Agile framework for managing and developing complex software, focusing on collaboration, adaptability, and iterative progress (see Figure 1.5). Unlike traditional methods, Scrum works in short, structured sprint cycles, typically lasting between one and four weeks, (more details are explained in **Appendix B**). Each sprint delivers a functional software increment, allowing teams to continuously refine their work based on feedback and evolving requirements [39]. This iterative approach makes Scrum effective in dynamic environments, such as sustainability-focused projects, where priorities can shift [28].
Scrum contributes to sustainability by promoting resource efficiency and waste reduction. Through practices like sprint planning and retrospectives, teams identify inefficiencies and streamline workflows, minimizing rework and optimizing the use of development resources

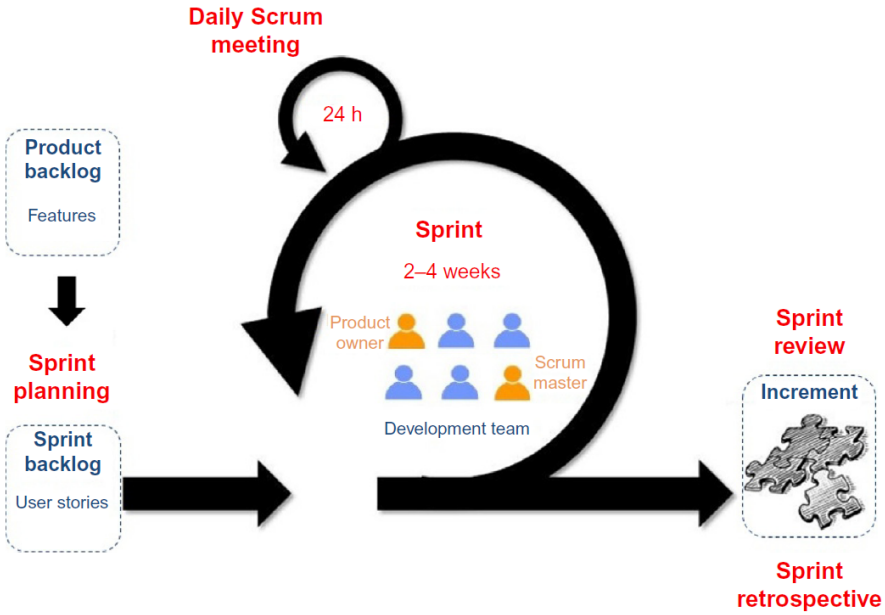


Fig. 7 The Scrum process.

Figure 1.5: Scrum Process

. By reducing software obsolescence and technical debt, Scrum also supports long-term sustainability. Integrating sustainability criteria into the development roadmap ensures that software remains maintainable and scalable, which aligns with broader sustainability goals by extending the software's life cycle and reducing its environmental impact [28].

1.4.4 Life Cycle Assessment

Life Cycle Assessment (LCA) is a method for evaluating the environmental impact of a product or process across its entire life cycle. In software development, LCA helps assess the environmental effects of the software product and its development process, from design through to end-of-life, considering both direct and indirect impacts [11, 18]. Figure 1.6 [6] shows how tools like SimaPro facilitate LCA by using data from unit processes stored in databases. This data is applied to a life cycle model to evaluate environmental impact. The results are then analyzed to understand the effects and guide more sustainable decision-making. By applying LCA to software engineering, it becomes possible to categorize, standardize, and prioritize environmental impacts, ensuring sustainability is integrated into design and process decisions, aligning software development with broader ecological goals [11].

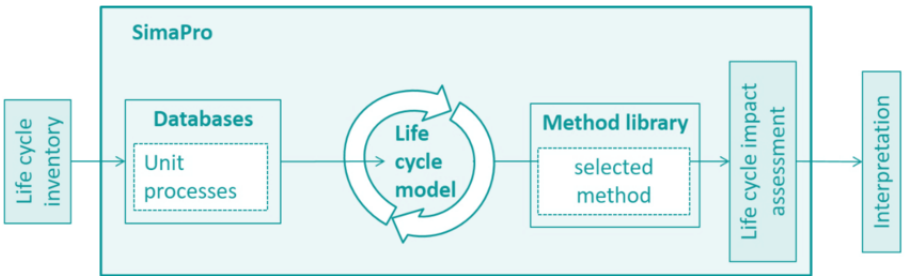


Figure 1.6: Use of SimaPro in LCA

Table 1.2: Aligning sustainability methods and their impact with SDLC phases.

Method	SDLC Phase	Sustainability Metrics	Purpose in Sustainability	Reference
Value Stream Mapping (VSM)	Analysis, Design, Implementation, Testing	Process efficiency, lead time reduction, and reduced rework	Identifies inefficiencies in workflows to minimise waste and optimise energy usage.	[43, 33, 24], [5], [4] [44]
Scrum	Design, Implementation, Testing, Deployment	Team velocity, reduction of wasted time, and client satisfaction	Ensures adaptive and efficient development cycles, reducing wasted effort and computing resources.	[8, 42, 35, 28]
Kanban	Implementation, Testing, Deployment, Maintenance	Flow efficiency, work-in-progress reduction, and operational stability	Limits work-in-progress to reduce unnecessary resource consumption and optimise workflow.	[39, 2, 15]
Life Cycle Assessment (LCA)	All Phases	Environmental impact, energy efficiency, carbon footprint	Evaluate software sustainability across all stages of the SDLC to minimise the ecological footprint.	[6, 21, 18]

Each tool contributes to sustainability in the SDLC by impacting key sustainability metrics. VSM improves process efficiency and lead time reduction, minimizing rework and energy consumption. Scrum helps reduce wasted time, increasing team velocity and client satisfaction while optimizing computational resources. Kanban improves flow efficiency, reduces WIP, and promotes operational stability, limiting unnecessary resource use. LCA provides a comprehensive view of the software's environmental impact, helping to reduce its carbon footprint and improve energy efficiency.
Table 1.2 maps these tools to relevant sustainability metrics, showing their direct impact on reducing waste, optimizing resource usage, and ensuring environmentally responsible software development.

In summary, integrating engineering techniques like VSM, Scrum, Kanban, and LCA into the SDLC helps optimize resource usage, reduce waste, and minimize environmental impact. By aligning these methodologies with key sustainability metrics, software development becomes more efficient, adaptable, and environmentally responsible.

1.5 Conclusion

This first chapter introduced the foundations of SSE, emphasizing the growing need to integrate sustainability principles into the SDLC. We examined how sustainability can be applied across environmental, economic, social, and technical dimensions, highlighting the importance of balancing performance, long-term viability, and responsible resource use. Finally, several methods were presented as potential enablers for embedding sustainability into different SDLC phases.

These theoretical foundations now serve as a reference for the rest of the report, which will analyze a real-world software development process and propose practical improvements based on the principles outlined here.

Chapter 2

Case Study : E-Commerce Platform Project

This chapter presents the case study used to apply the sustainable software engineering principles introduced earlier. The project focuses on building an e-commerce platform for a local men's fashion brand in Algeria, developed under the supervision of ICOSNET, a leading IT services company.

The chapter begins by presenting the departement structure and the development team allocated for the project, then analyzing ICOSNET's current development model, which follows a traditional Waterfall approach, it then describes the project context, objectives, and implementation challenges. . Each phase of the SDLC is examined to understand its structure, deliverables, and limitations in terms of adaptability and sustainability. Diagnostic tools such as SWOT, Ishikawa, and the 5 Whys are used to identify inefficiencies and root causes of recurring problems.
Finally, a comparative analysis of potential sustainable methods is conducted. A weighted multicriteria matrix is used to evaluate techniques like Scrum, Kanban, VSM, and LCA based on their feasibility, efficiency, and alignment with sustainability goals. The chapter concludes by selecting the most appropriate method to simulate an optimized SDLC in the next chapter.

2.1 Presentation of the Internship Hosting Company

Icosnet, founded in 1999, is a leading telecommunications and IT solutions provider in Algeria. As the country's sole private Internet Service Provider (ISP), Icosnet delivers a broad range of services to enterprises, SMEs, and international clients. The company operates two Tier III-certified data centers in Algiers and Oran, ensuring high availability and compliance with national data regulations.
Icosnet's service offerings are organized into six main categories (see Figure 2.1): **Internet Access Solutions, Cloud Services, Cybersecurity Solutions, Unified Communications, Data Center Infrastructure, and E-commerce Solutions**. The company is committed to innovation, security, and scalability, working with global technology leaders like VMware, Sophos, and IceWarp. Through its AI-driven cybersecurity systems and disaster recovery plans, Icosnet continues to drive Algeria's digital transformation and meet the evolving needs of its clients across various sectors.

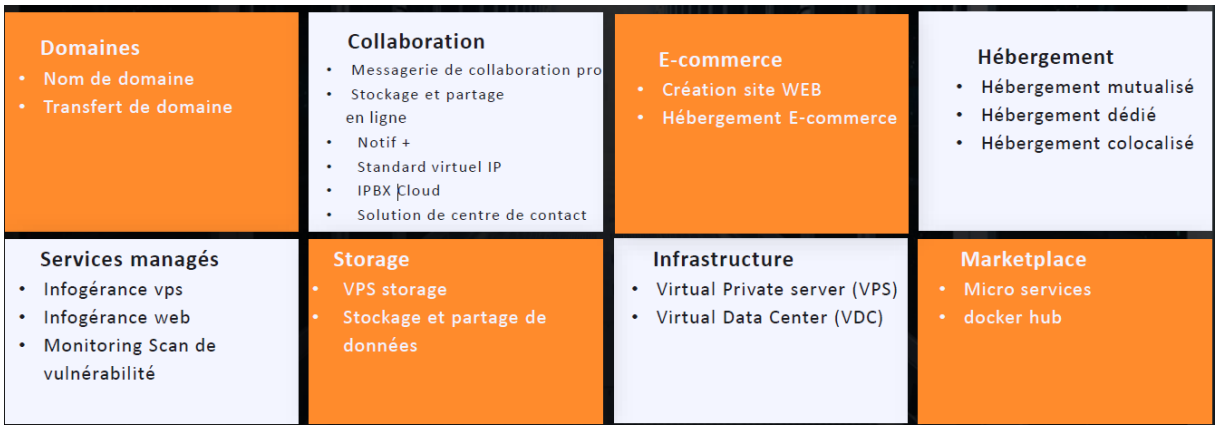


Figure 2.1: ICOSNET's Solutions (internal document)

2.2 Development Team and Department Structure

The development of the e-commerce platform was carried out within the Development Department of ICOSNET, operating under the supervision of the **Chief Technical Officer (CTO)**. As part of the company's broader technical operations, this department focuses on software engineering, web platform development, and technical innovation in alignment with the digital transformation strategy.
For this project, a dedicated five-member team was assembled, The team included:

- **Project Manager / Technical Lead** (dual role)
- **Three Full-Stack Developers**
- **One QA Tester / Designer** (shared responsibilities)

This compact but versatile team structure enabled full-cycle development, from frontend and backend implementation to UI/UX refinement and manual testing. The Project Manager also acted as the Technical Lead, ensuring both strategic alignment with project goals and direct oversight of code quality and architecture decisions.
The Development Department itself is one of the key units under the CTO's leadership and collaborates closely with other technical divisions such as infrastructure, cybersecurity, and cloud services. Project execution also involves interaction with supporting roles including the Project Manager Officer, Quality Manager, and Internal Audit teams, ensuring cross-functional alignment with company-wide compliance and performance standards.

Figure 2.2 shows the organizational hierarchy of the internship hosting company.

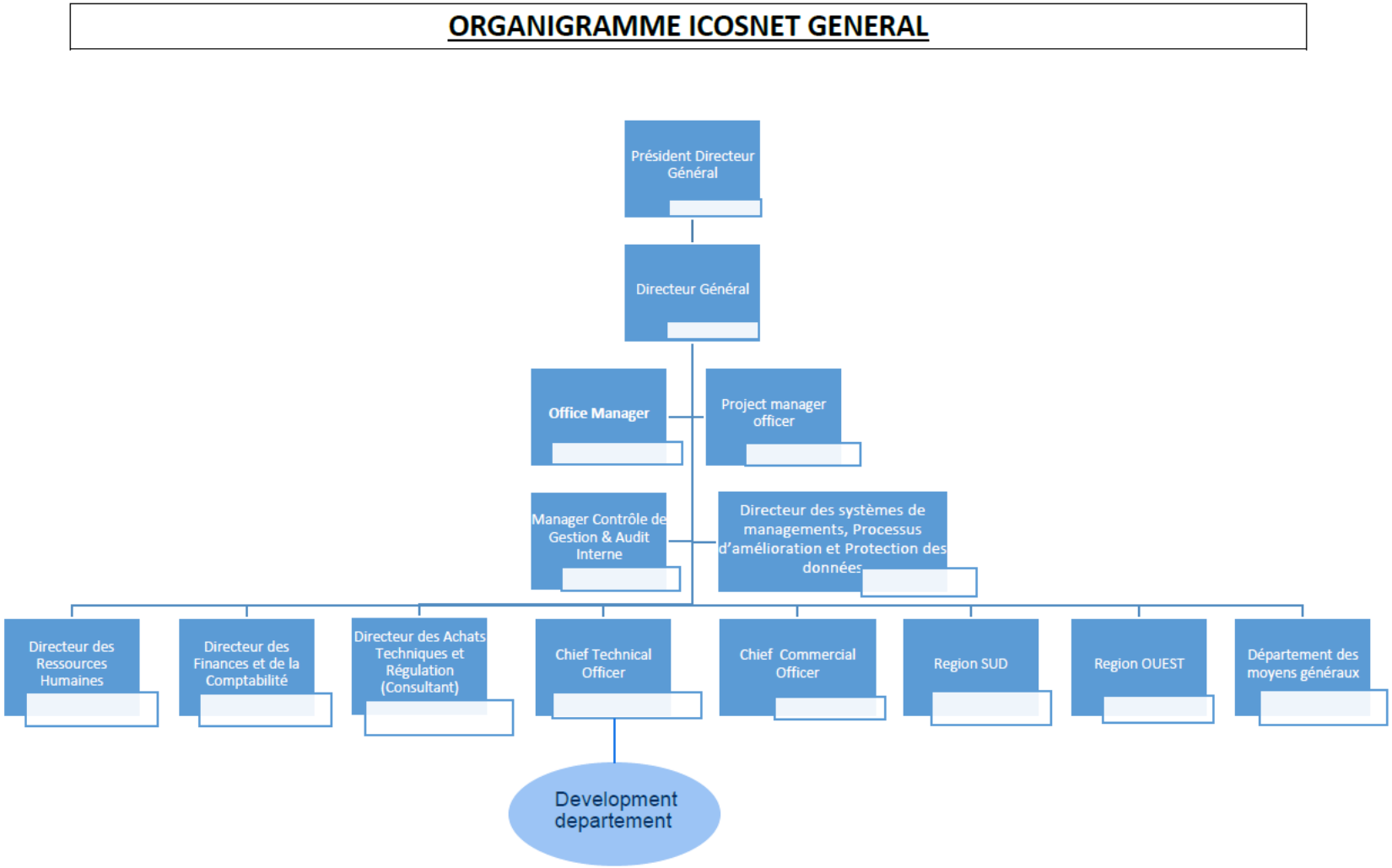


Figure 2.2: ICOSNET's Organizational Hierarchy

2.3 Project overview

This project involves developing a dedicated online store for a local men's fashion brand seeking to expand into digital commerce. The platform is designed to address specific challenges in the Algerian market like payment integration and delivery logistics, while providing a smooth, locally adapted shopping experience. To better understand the context and scope, the table 2.1 summarizes the key elements of the project. For a full strategic view, the Business Model Canvas (BMC) is presented in **Appendix A**.

Table 2.1: E-Commerce Project Overview

Aspect	**Description**
Project Title	E-Commerce Platform for a Local Men's Fashion Brand
Client	Fashion retailer expanding into digital commerce (via ICOSNET service offering)
Target Market	Algerian men, primarily young adults seeking modern, affordable style
Scope	Full-featured online store with admin tools, payment integration, and delivery support
Development Technology	• Modern web stack (React) with responsive design • Node.js for secure order management and database connectivity
Key Features	Product catalog, checkout, payment (CIB, BaridiMob, COD), order tracking, admin dashboard
Deployment Environment	Hosted via ICOSNET's infrastructure.
Expected Outcome	A scalable, user-friendly e-commerce platform tailored to local consumers habits

2.4 Current SDLC Model

This section presents a structured analysis of the software development process currently used by the team. The team follows a traditional **waterfall model** [36], where development flows sequentially from one phase to the next. Each stage of the SDLC is mapped and reviewed with its corresponding practices and deliverables. Alongside this breakdown, we assess how well the current process aligns with sustainability principles and highlight key inefficiencies.

This model offers a clear structure from start to finish, with each phase handled by specialized roles. It helps manage tasks, deadlines, and responsibilities, keeping the work organized and reducing risk throughout the project [16].

The SDLC at ICOSNET includes six main phases: Analysis, Design, Implementation, Testing, Deployment, and Maintenance. Each step has its own goals and outputs, and the team moves from one phase to the next in a sequential way [10]. While some small adjustments are made depending on the project, this structure stays mostly the same

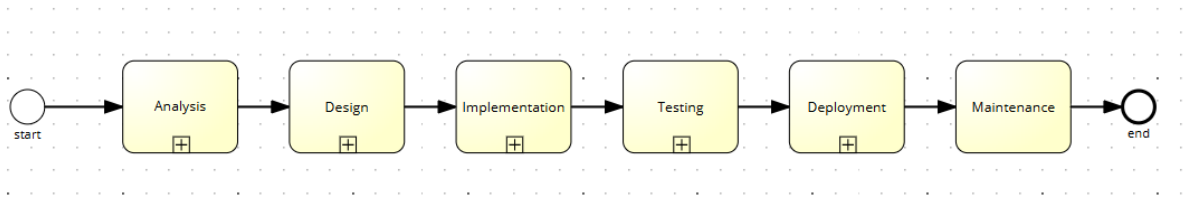


Figure 2.3: High Level SDLC Model

across all developments. The figure 2.3 shows the general SDLC model currently applied at the company.

2.4.1 Current Process – Requirement Analysis Phase

The analysis phase is the first step in the development process. It focuses on understanding the client's needs and translating them into clear, structured requirements.

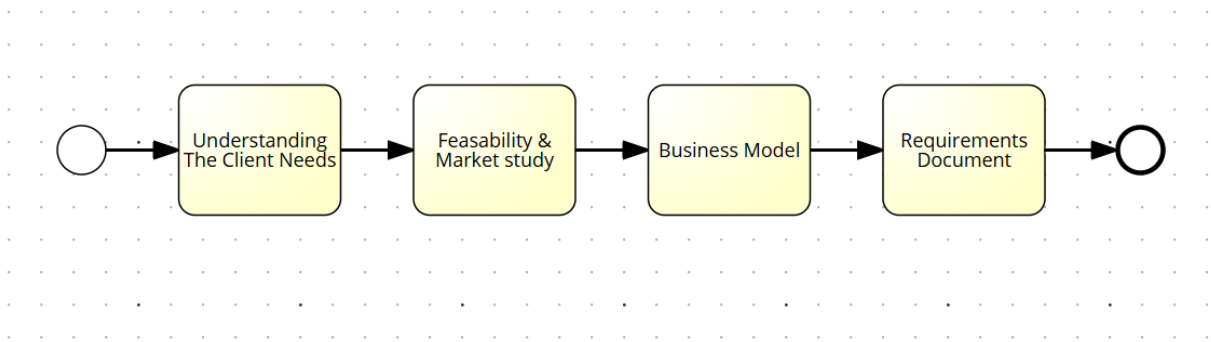


Figure 2.4: Requirements Analysis Phase

It begins with meetings between the client and the internal team to define the project's main objectives, target users, and expected functionalities. This early exchange helps align the project with both technical capabilities and business goals.
For market-oriented projects like e-commerce, a quick analysis of local trends is also done. This helps the team understand user preferences, especially regarding payments, delivery, and platform usage. The findings help shape a realistic and user-focused solution from the start.
The main output is a formal requirements specification document. It includes functional and non-functional needs, use cases, roles, and constraints. However, this process remains traditional, with limited client feedback once the document is finalized. Figure 2.4 shows how this phase is currently structured at ICOSNET.

2.4.2 Current Process – Design Phase

Once the requirement document is approved, the design phase begins. This step translates client needs into a technical blueprint through collaboration between developers and designers. It starts with high-level system design, where the architecture, data flow, and main modules are outlined.
The UX/UI team then creates wireframes and mockups using tools like Figma. These visuals guide the user experience and are reviewed by the project manager. In parallel, the technical stack is defined based on performance needs and team expertise. Technologies

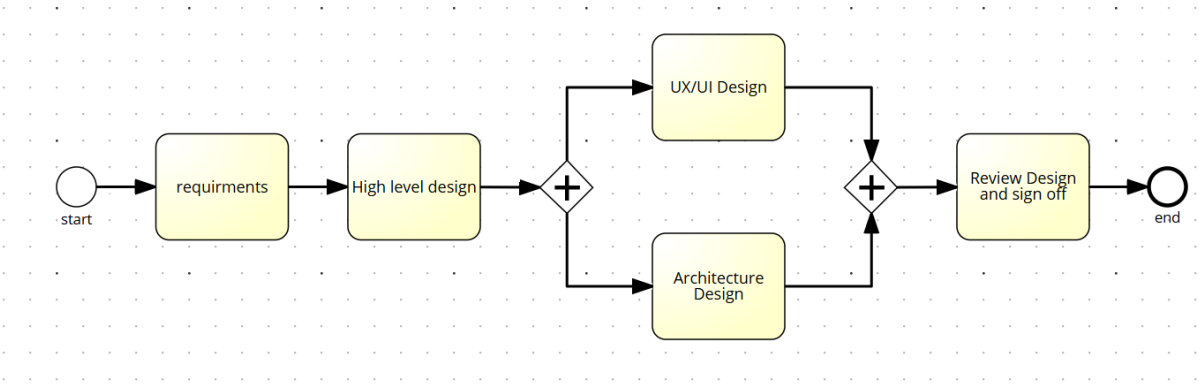


Figure 2.5: Design Phase Process

like React, Node.js, or Laravel are often selected. Database structures and APIs are also designed to ensure integration between frontend and backend.
Before implementation starts, a formal design review is held to align the team and validate the plan. Figure 2.5 illustrates this process. While the approach is solid, sustainability factors, such as energy efficiency or long-term maintainability, are not yet considered, highlighting a potential area for improvement.

2.4.3 Current Process – Implementation Phase

The implementation phase follows a structured workflow that begins with task delegation and ends with code review. Tasks or tickets are assigned by the project manager based on the design specifications. Developers then proceed to write code for the assigned modules, using Git to manage changes and ensure collaboration.

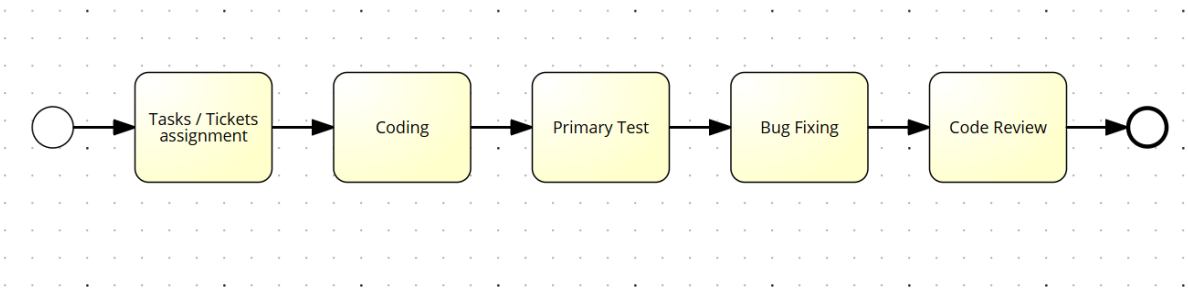


Figure 2.6: Implementation Phase Process

Once development is complete, the developer performs initial manual testing to verify that the functionality works as expected. This testing is typically done by the same developer who wrote the code and is mostly manual. If issues arise, a bug fixing cycle begins, where the same developer iterates until the issue is resolved. After basic validation, the code is submitted for review. The team members check for logic, performance, and adherence to standards before approving the merge.
This disciplined approach ensures quality control, as shown in Figure 2.6. However, sustainability aspects such as energy-efficient coding or hardware resource usage are not considered during development or review.

2.4.4 Current Process – Testing Phase

Testing begins after the main development work is complete.

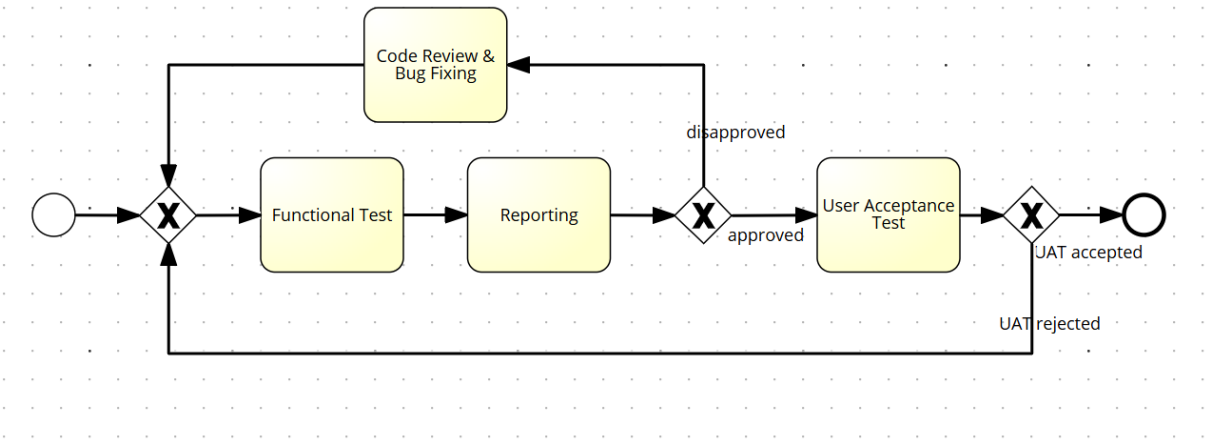


Figure 2.7: Testing Phase Process

The QA engineer manually checks the platform's features based on the requirements document. However, there's no formal test plan early on, the test cases are usually written late, just before testing starts. This leads to last-minute issues that sometimes delay the project. Bugs are tracked using basic tools and fixed in cycles, with the code sent back for retesting. The only structured client feedback happens during the **User Acceptance Testing**. At this stage, the platform is shared with the client for review. They test key features and give feedback, but since this happens late in the cycle, most suggestions are either postponed or cause delays if they require major changes.
As shown in Figure 2.7, the testing process works, but it's slow and reactive. Without automation or long-term quality strategies, there's little room for efficiency or sustainability improvements.

2.4.5 Current Process – Deployment Phase

Once the application passes both internal and client validation, the team moves to production setup. This includes configuring servers, adding SSL certificates, and preparing the database and API connections. Everything is done manually, there are no automated deployment pipelines. While this gives developers more control, it also increases the risk of errors or delays if someone is unavailable.
A final round of live testing is done to check critical functions like login and checkout. If all works well, the platform is deployed in a single version. In case of serious issues after launch, the team uses manual rollback, as there's no automated backup or backup system in place.
As shown in Figure 2.8, the process is straightforward but lacks automation and monitoring. Without tools for resource optimization or green hosting, the platform may also use more energy than needed.

2.4.6 Current Process – Maintenance Phase

In the current SDLC model, the maintenance phase is managed by the same development team responsible for the platform's implementation. When a client reports a problem :

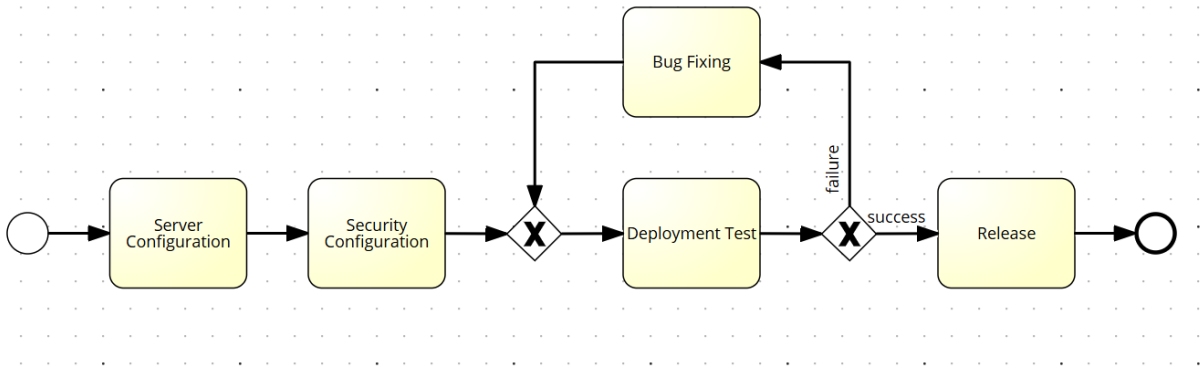


Figure 2.8: Deployment Phase Process

a server downfall, a critical bug, or an unexpected behavior,the team immediately delays ongoing project work to address what they call *les quotidiens* (daily incidents). These interventions are treated as top priority and are resolved as quickly as possible to restore service continuity. This introduces challenges in terms of workflow management and performance evaluation. Since team members shift unpredictably between development tasks and incident resolution, it becomes difficult to measure individual productivity. There is no systematic way to differentiate time spent on client support from time allocated to planned project tasks.
This model focuses only on keeping the platform operational, there is no structured feedback loop, no usage analytics, and no plan for continuous improvement unless explicitly requested by the client. Moreover, the absence of a dedicated maintenance team results in context-switching, which can reduce code quality and increase the risk of technical debt over time.

The current software development process follows a structured and sequential waterfall model, progressing through six main phases: analysis, design, implementation, testing, deployment, and maintenance. Each phase is clearly defined and executed by the internal team based on established practices. This approach ensures technical reliability but lacks iterative feedback loops and adaptability.
In the next section, we will evaluate this model through diagnostic tools to better understand where inefficiencies occur and how improvements can be made from a sustainability point of view.

2.5 Diagnostics of the Current Software Development Process

This section builds on the previous analysis of the current SDLC by applying diagnostic tools to highlight inefficiencies, risks, and missed opportunities related to sustainability. While the existing SDLC is stable and well-structured, it lacks flexibility and does not fully align with sustainability goals, particularly in areas like energy efficiency, resource optimization, and client collaboration.
To better understand the root causes of these limitations, three well-established analysis methods are applied:

- **SWOT Analysis**, to assess internal strengths and weaknesses as well as external opportunities and threats.

- **Fishbone Diagram (Ishikawa)**, to visualize the underlying causes behind key sustainability challenges.
- **5 Whys**, to break down specific issues related to rework, lack of feedback, and inefficient practices.

2.5.1 SWOT Analysis

The SWOT matrix is a strategic analysis tool used to identify internal and external factors that affect a project or organization. It helps evaluate Strengths, Weaknesses, Opportunities, and Threats to provide a clear overview of current conditions. In the context of this software development process, it was used to highlight both technical and organizational aspects that influence performance, sustainability, and client satisfaction.

Figure 2.9 presents the SWOT matrix applied to the current development process.

The SWOT analysis reveals a development environment with a strong technical foundation but notable gaps in process agility and sustainability integration. On the internal side, strengths include a skilled development team, use of modern development technologies, and access to the secure hosting infrastructure. These factors provide a solid base for high-quality software delivery.
However, several weaknesses affect long-term efficiency and adaptability. The absence of sustainability criteria, lack of KPI tracking, and reliance on manual workflows limit the team's ability to measure and improve performance. Additionally, feedback from clients is only collected at the final stages, increasing the risk of misaligned features and costly rework.
The opportunities identified point toward the growing demand for local e-commerce platforms in Algeria and the potential to integrate lighter, greener features (e.g., image caching, modular APIs) to improve energy efficiency. These trends align with broader sustainability goals and represent strategic directions for improvement.
On the other hand, threats include increased competition in the digital services market and the risk of client dissatisfaction due to late delivery adjustments. Without proactive changes, these threats could undermine project outcomes and business reputation.

Overall, the SWOT matrix supports the need for a more iterative and metrics-driven solution, making Scrum a strong candidate for addressing the identified weaknesses and seizing available opportunities.

2.5.2 Ishikawa Diagram

The Ishikawa diagram, also known as the fishbone diagram, is used to identify the root causes behind a recurring problem or inefficiency. In our case, it helped categorize the different factors affecting sustainability and development quality. These factors were grouped under categories like Method, Manpower, Milieu (Environment), Materials, and Measurements to provide a clear visual breakdown of potential issues.

Figure 2.10 illustrates the Ishikawa diagram for the current development process.

The central study focus was **The Development Process** referring to issues such as rework, low adaptability, and lack of process visibility.

The diagram organizes the causes into five main categories: Milieu (Environment), Manpower, Method, Material, and Measurements.

- **Milieu**: Highlights environmental distractions, such as lack of quiet space for focused development, frequent interruptions, and poor internal communication.
- **Manpower:** Reflects a small and overstretched team, with developers handling both development and client support, skill gaps and the absence of formal retrospectives further reduce learning.
- **Method:** Points to the linear, phase-based SDLC model and the lack of mechanisms for continuous improvement, sustainability integration, or structured client feedback loops.
- **Material:** Identifies technical and tooling limitations, including reliance on spreadsheets, unreliable testing environments, and hardware issues affecting productivity.
- **Measurements:** Exposes the complete absence of performance tracking—no KPIs for code quality, test coverage, or developer effort—making it difficult to assess or improve over time.

This analysis reveals how deeply interrelated and systemic the inefficiencies are. Many of the problems trace back to structural and methodological limitations, not just technical constraints. The diagram also illustrates how the lack of sustainability practices, weak internal coordination, and the overload of multitasking developers compound each other to affect delivery quality and team well-being.
By visualizing these root causes, the Ishikawa diagram strongly supports the case for adopting Agile methods as Scrum, to bring structure, metrics, and focus to the development process. It also highlights the need for dedicated roles, better tooling, and environmental adjustments to improve both productivity and sustainability outcomes.

2.5.3 The 5 Whys Method

The 5 Whys is a root cause analysis tool that helps dig deeper into a specific issue by repeatedly asking "why" until the underlying problem is identified. In this study, it was applied to two main issues observed during the internship: rework due to late bug discovery, and lack of traceability due to the absence of project management tools. This method revealed structural limitations of the current SDLC that affect efficiency and quality.

Figures 2.11 present the two 5 Whys diagrams used in this analysis.

In both cases, the analysis led to the same root cause, which is the rigidity of the Waterfall model, which lacks intermediate feedback loops and early testing mechanisms. Feedback is typically collected only during UAT, making it too late to influence design decisions without costly rework. Similarly, testing occurs at the end of the cycle, allowing undetected issues to accumulate. These problems are not caused by technical shortcomings, but by a linear process model that prevents adaptation and continuous improvement. From a sustainability perspective, this structure blocks the monitoring of quality, efficiency, and stakeholder alignment throughout the project. The findings confirm the need for a more iterative and flexible approach like Scrum, where feedback is built into every stage of development.

2.5.4 Diagnostic Results and Improvement Proposals

The combined diagnostic results reveal a development process that, while operationally functional, suffers from rigidity, poor visibility, and a lack of sustainable practices. All three tools point to recurring issues such as delayed feedback, rework, missing performance metrics, and outdated tools. The SWOT matrix highlighted strategic weaknesses like the absence of KPIs and sustainability criteria, the Ishikawa diagram traced inefficiencies to staffing, tooling, and process gaps, and the 5 Whys confirmed that the root cause lies in the linear SDLC model. These findings collectively demonstrate that the current workflow does not suite the dynamic needs of modern software projects, reinforcing the need for a more iterative, measurable, and client-centered approach.

To clearly synthesize these observations, Table 2.2 links the main anomalies to their root causes and outlines targeted solutions. It serves as a practical roadmap for transforming the current development approach into one that aligns with SSE principles. The recommended changes target the pain points od the development process.

Table 2.2: Observed Issues, Root Causes, and Improvement Proposals

Observed Issue	**Root Cause**	**Suggested Improvement**
Delayed client feedback	Linear SDLC; feedback occurs only at UAT	Adopt Scrum with regular sprint reviews to involve clients earlier
Rework and late bug discovery	Manual testing and late validation cycles	Integrate continuous testing and test automation in each sprint
Lack of performance visibility	No KPIs, no tracking tools	Introduce simple metrics (velocity, rework hours) and visual dashboards
Unpredictable workload during maintenance	Shared resources and informal prioritization	Define a support buffer within each sprint; separate incident handling
Tooling gaps (manual tracking, outdated processes)	No digital PM tools; limited process evolution	Introduce lightweight tools (e.g., Trello, Jira) and Agile estimation techniques
No sustainability metrics	Sustainability not embedded in process or goals	Map SSE indicators (e.g., energy usage, tech debt) to development activities

These findings make a compelling case for adopting a structured yet flexible approach like Scrum, supported by SSE principles. Scrum enables iterative delivery, continuous client feedback, and measurable progress, while SSE adds a long-term perspective by encouraging maintainability, energy efficiency, and stakeholder-centered outcomes.

Figure 2.9: SWOT matrix applied to the current development process.

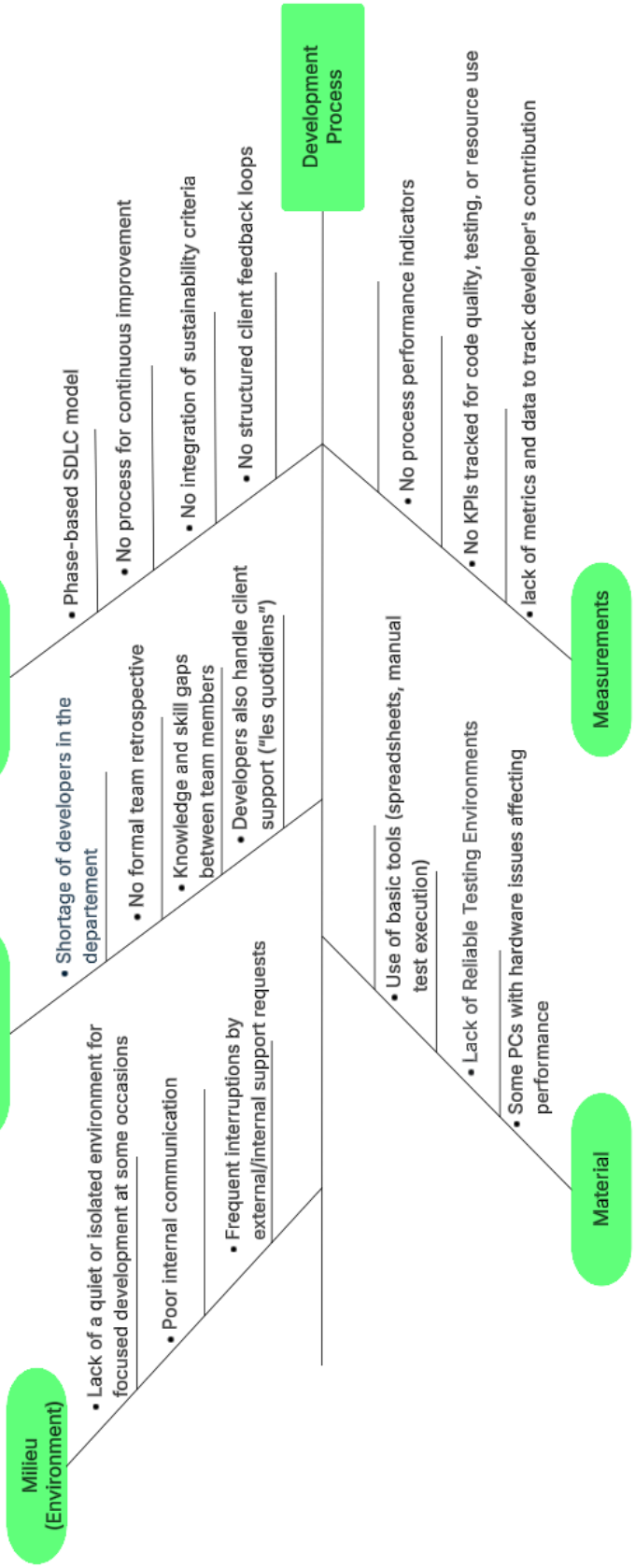


Figure 2.10: the Ishikawa Diagram for the Current Development Process

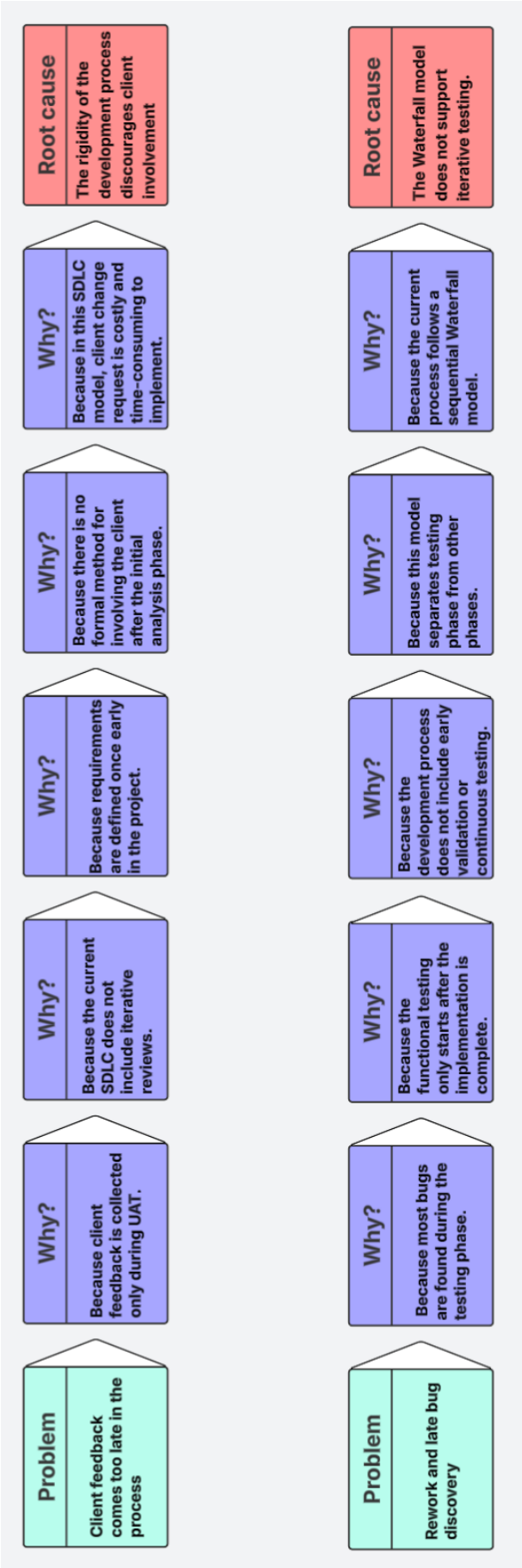


Figure 2.11: The Two 5 Whys Diagrams Used in Diagnostic Analysis

2.6 Methodological Selection Approach

After identifying the limitations of the current SDLC model, this section focuses on selecting the most appropriate methods to optimize the process. The goal is to adopt a structured, sustainability-aware development approach that better addresses the challenges highlighted in the previous diagnostics.
To support this transition, a multi-criteria decision-making process was used. The approach begins with defining six weighted evaluation criteria based on both practical feasibility and sustainability priorities. These criteria are then used to compare four candidate methods: Scrum, Kanban, Value Stream Mapping, and Life Cycle Assessment.
Each method is scored based on expert input and field relevance, resulting in a weighted matrix that guides the final selection. This structured framework ensures that the chosen methodology is well-aligned with project needs, team capabilities, and SSE objectives.

2.6.1 Criteria for Method Selection

To identify the most suitable methods for our project's sustainable development simulation, a set of six evaluation criteria was selected. These reflect a combination of technical feasibility, sustainability objectives, and organizational priorities. The weight assigned to each factor represents its strategic importance for ensuring both project success and long-term impact. These weightings were discussed and validated with the project tutor and department leads at Icosnet to ensure alignment with internal expectations and real-world constraints.

- **Feasibility (30%)**
  Feasibility carries the most weight, nearly one-third of the decision matrix, because no method, no matter how sustainable in theory, is useful if it cannot be realistically implemented. It covers aspects like ease of integration, availability of data, team familiarity, and tool readiness. In a context like ours : limited timeframe, fixed team structure, and predefined tech stack, this becomes the most limiting factor. *The Chaos Report* [14] confirms this by linking project failures to unrealistic planning and overly ambitious scope. Similarly, *Gärtner and Eriksson's* case study [19] showed that even experienced teams struggle with applying advanced practices like Continuous Delivery unless properly coached and supported. Tools reviewed in *Gartner's Magic Quadrant* [31] often cite "ease of adoption" as a competitive edge. All of this reinforces why feasibility needs to drive the method selection.

- **Resource Efficiency (20%)**
  This factor is the second highest because of its double impact: better use of developer time and infrastructure, and reduced chances of rework or bottlenecks. A method that helps avoid redundant tasks or manual tracking can significantly improve delivery speed without increasing energy consumption or workload. In some reviews, over 50% of sustainable success factors were tied to resource efficiency like "paperless communication" and lean documentation [40]. Others also connect efficient algorithms and reuse practices to both sustainability and productivity [30]. And *the standish group* [14] reveals that poor resource use contributes to the 189% average cost overruns seen in challenged projects. In that sense, resource-efficient methods don't just save effort, they save the project.

- **Environmental Impact (15%)**
  Although environmental impact is central to sustainable software engineering, its relative weight is moderate. This reflects a practical consideration: not all methods have direct or immediate effects on energy use or emissions, and in short-term simulations, these impacts can be harder to quantify. Studies stress the value of methods that promote energy-efficient design, low-carbon code, and reduced infrastructure demands. When sustainability becomes embedded early in the process, even small gains like smarter caching or reduced builds, can lower power consumption and increase overall eco-performance [40, 30].

- **Client Feedback Integration (10%)**
  While not as heavily weighted as feasibility or efficiency, client feedback remains a vital component of sustainable and adaptive development. It helps avoid misaligned work and reduces the risk of late-stage revisions that consume additional time and energy. According to the *Chaos Report* [14], user involvement is the top-ranked success factor across all project types. The faster teams can validate work with the client, the fewer unnecessary iterations they'll face. In practice, this means a method that encourages regular feedback has a clear edge in terms of agility and responsiveness.

- **Scalability & Reusability (15%)**
  Reusability supports long-term sustainability by reducing duplication and increasing maintainability. A reusable component today saves development time tomorrow. At the same time, scalability ensures that methods and processes can grow with the product or be reapplied to similar projects. The literature explicitly lists reusability as a sustainability success factor, while *Mahmoud & Ahmad* [30] integrate it at both the design and end-of-life stages of their model. It also shows that scalable approaches lead to more stable delivery cycles and smoother team onboarding, which adds long-term value even if the immediate benefits aren't always visible [40, 35].

- **Team Alignment & Collaboration (10%)**
  Collaboration is critical for method adoption and team morale. Methods that support shared understanding, clear roles, and transparent progress are more likely to stick and lead to successful delivery. In *Gärtner et al.* case study [19], the team's success in applying a complex delivery pipeline was credited largely to collaborative practices like stand-ups, retrospectives, and informal check-ins. Tools that support real-time collaboration also reduce noise and increase team cohesion. When a method promotes a healthy, communicative environment, it indirectly improves all other dimensions.

Validation with experts:
The chosen criteria and weights were discussed during internal meetings and validated by the project supervisor, PMO, and department leads. In addition, the scoring of each method, based on feasibility, resource efficiency, and other factors, was reviewed and approved by the same experts, ensuring the chosen approach aligns with both technical priorities and organizational realities.

2.6.2 Multicriteria Decision Matrix

The matrix provides a structured comparison, ensuring that the final choice is supported by both practical relevance and alignment with sustainable development goals. The results of this evaluation are presented in the table 2.3 below:

Table 2.3: Multicriteria Decision Matrix

Criteria	**Weight**	**Scrum**	**Kanban**	**VSM**	**LCA**
Feasibility	0.30	4	4	3	2
Resource Efficiency	0.20	4	4	5	3
Environmental Impact	0.15	3	3	4	5
Client Feedback Integration	0.10	5	3	2	1
Scalability & Reusability	0.15	4	4	3	2
Team Velocity & Collaboration	0.10	5	4	3	2
Weighted Score		**4.05**	**3.75**	**3.45**	**2.55**

2.6.3 Final Selection Based on Decision Matrix

The multi-criteria matrix evaluated Scrum, Kanban, VSM, and LCA based on key sustainability-related criteria, with Feasibility and Resource Efficiency given the highest weight. These priorities reflect the issues previously identified through the SWOT, Ishikawa, and 5 Whys analyses, including poor feedback loops, lack of KPIs, and rigid planning models.

Scrum scored the highest **(4.05/5)**, mainly due to its strong support for iterative feedback and collaboration. These strengths directly address the process rigidity and late client involvement observed in the diagnosis.
Kanban **(3.75/5)** ranked second. It promotes continuous delivery and resource control but lacks the structured planning and stakeholder feedback that Scrum offers.
VSM received a **(3.45/5)**. It helps identify inefficiencies and environmental waste but is harder to apply consistently due to data and implementation gaps.
LCA scored the lowest **(2.55/5)**, confirming that While it excels in environmental impact assessment, its practical application in development processes remains challenging. It remains a valuable supporting tool, particularly in design and deployment phases.

The diagnosis and the matrix supports our decision to **adopt Scrum** as the main framework for the process optimization, with the highest score of 4.05, Scrum offers the best balance across all selected criteria. It stands out in areas such as client feedback integration, team collaboration, and scalability, all of which are essential for delivering a sustainable, iterative software process. Its strong feasibility and alignment with agile principles.

2.7 Conclusion

The diagnostic tools applied in this chapter exposed clear weaknesses, including the absence of KPIs, limited client engagement during development, and a lack of tools for process tracking and automation. These factors contribute to performance bottlenecks and resource waste, while also hindering the team's ability to iterate or make data-driven

improvements. On the other hand, the project's strengths like the well-defined goal and modern tech stack offer a strong foundation for improvement. The findings highlight a need to transition toward a more adaptive and collaborative development process that integrates sustainability as a core consideration.

Chapter 3

Scrum Implementation in the SDLC

This chapter introduces the optimized SDLC adopted for the e-commerce platform project, following the diagnostic findings of Chapter 2. The goal is to improve the development process previously in use for a more flexible, iterative framework that better aligns with both the project's objectives and sustainability principles.

To do so, Scrum was selected as the core methodology to simulate and structure the Design, Implementation, Testing, and Deployment phases as one continuous agile cycle. This new flow offers enhanced responsiveness, early feedback, reduced rework, all while improving operational efficiency.

3.1 Scrum Framework Overview

Scrum was adopted as the foundation of the new SDLC. It introduces structured roles, recurring events, and essential artifacts that guide iterative development and team collaboration. A full overview of the Scrum framework, including roles, artifacts, events, and a comparison with other approaches, is presented in the cheat sheet (**Appendix B**) to support better understanding of the work.

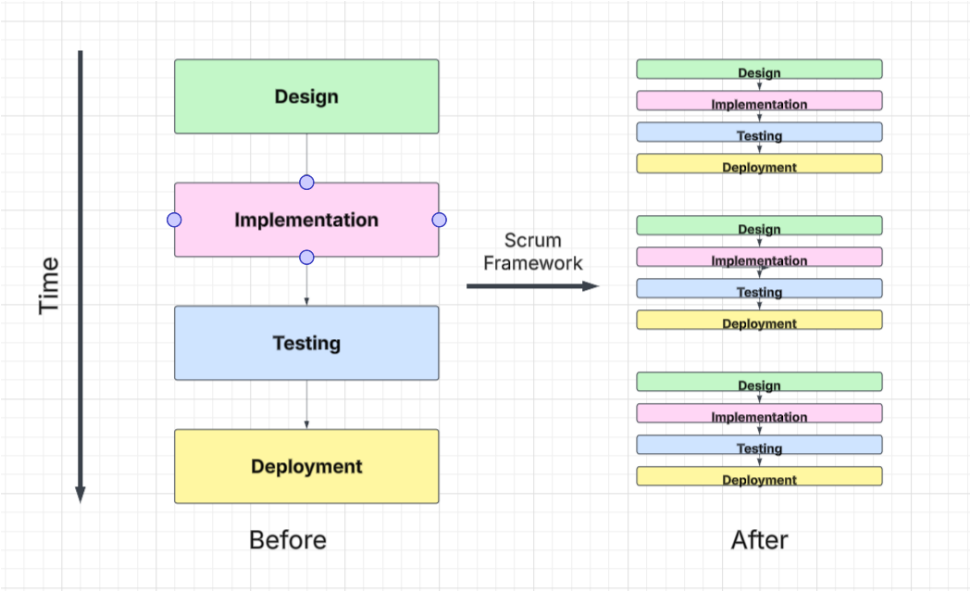


Figure 3.1: Transition to Scrum

3.2 The Proposed SDLC

Figure 3.1 illustrates the transition to the newly proposed Scrum-based workflow.

Initially, the development process caused delays in issue detection, limited client feedback to the very end, and led to increased rework and inefficient resource use.
The process was redesigned using an Agile simulation that integrates Scrum practices across the four central SDLC phases. Each sprint now includes a full cycle of design, coding, testing, and deployment tasks. This reorganization fosters continuous feedback, early validation, and a more sustainable workload. It also improves collaboration, reduces delays, and aligns the team's output with evolving client needs.

3.3 Project Simulation with Scrum

This section presents the practical application of the optimized SDLC using the Scrum framework.

Figure 3.2: Scrum Implementation Plan

The implementation is structured into three connected steps, as illustrated in Figure 3.2, each step represents a key phase of the project simulation:

- **Step 1:** Team Setup and Planning : presenting the team structure, the sprint planning logic, and backlog estimation strategy.
- **Step 2:** Sprint Execution : showing how development, testing, and client validation unfolded during the simulation.
- **Step 3:** Continuous Improvement : summarizing feedback loops, process adjustments, and sustainability-related observations for future sprints.

The goal is to demonstrate how the integration of Scrum supports better task management, faster feedback, and improved sustainability practices across the design, implementation, testing, and deployment phases.

3.3.1 Step 1: Setup and Team Alignment

The first step in applying Scrum to the project was to define the roles and prepare the team to adopt an iterative workflow. The setup was done in a simplified and realistic way, keeping in mind the internal structure and constraints of the organization.

*See **Appendix C** for the official Scrum role definitions.*

Initial Findings:

Table 3.1: Team Roles and Responsibilities

Role	Assigned To	Responsibilities
Product Owner	Project Manager + PMO	Prioritized the backlog, validated features, and represented business needs
Scrum Master	External IT Consultant	Facilitated events, removed blockers, and supported Agile discipline
Scrum Team	3 developers + QA tester	Handled full-stack tasks flexibly; no strict handovers needed

- A kickoff meeting was held with the Product Owner, PMO, and Scrum Master to align expectations.
- User stories were not written in the standard "As a user, I want to..." format, but the backlog items were functional and clear enough to serve as the basis for early sprint work.
- Tools such as Jira were not used, but an old spreadsheet from a previous project was repurposed to track tasks and simulate the Scrum board (see Figure 3.3).
- Task estimation was still informal.

This phase laid the groundwork for the project's transition to Scrum. While not every practice was implemented by the book, the team managed to set up a working structure that balanced organizational habits with Agile principles, making the upcoming simulation both functional and realistic.

3.3.2 Step 2 : Sprint Execution and Feature Prioritization

After setting up the Scrum team, the project moved into a two-week sprint. The goal was to test delivery rhythm, validate backlog-driven planning, and build a smoother, iterative workflow.

Backlog Prioritization with MoSCoW

Before sprint planning, the existing backlog was reviewed using the MoSCoW technique [26, 22] to help structure the work based on client needs and technical feasibility:

Table 3.2: MoSCoW Prioritization of Project Requirements

Category	Example Requirements
Must Have	Product catalog management, local payment integration (Edahabia, CIB), admin dashboard
Should Have	Order tracking system, secure checkout, basic analytics
Could Have	Loyalty program, customer reviews, promotional pop-ups
Won't Have	AI-based recommendations, advanced CRM tools

This prioritization helped define a realistic **Sprint Goal** and filtered high-value features for early delivery.

A	B	C	D	E	F	G	H	I	J	K	L
Story ID	**Story Name**	**As a... (Actor)**	**I can... (Goal)**	**Status**	**Priority**	**Dependencies**	**Story Points**	**Dev Increment**	**Dev Owner**	**Dev Complete Date**	**Actual Dev Time (man days)**
S073	View Assessment Summary	Organisation User	View summary data about my assessment	Implement	Should		6	v8.0 (i4)	NL	21/04/2025	7
S136	View Assessment Summary Target Score	Organisation User	View my target score along with my assessment summary	Implement	Should		1	v8.0 (i5)	NL	28/04/2025	1
S077	Look and Feel	General User	Use the application easily and without aesthetic insult to my eye	Implement	Must		6	v8.0 (i2)	TM	25/02/2025	4
S104	Look and Feel v2	General User	Have an enhanced aesthetic experience, with lovely soft curves caressing my visual senses	Implement	Must	S077	5	v8.0 (i2)	TM	04/03/2025	4
S078	Improved Navigation	General User	Navigate the site more easily	Implement	Must		2	v8.0 (i2)	DES	18/02/2025	2
S096	Improved Navigation - Information Mapping	Information Mapping User	Be able to navigate to the information mapping functions	Implement	Must	S078	1	v8.0 (i3)	NL	19/03/2025	0,5
S074	Navigate to Views Maintenance	XYZ ABC User, Organisation Administrator	Be able to access the Views Maintenance function	Withdrawn	Could			N/A			
S079	Home Page	General User	View information on the home page	Implement	Must		2	v8.0 (i2)	TM	24/02/2025	2

Figure 3.3: The Product Backlog

Sprint Setup and Planning

Based on the product backlog, the PO selected a set of stories that reflected a mix of functional importance and technical feasibility. Each story followed a basic *"As a... I can..."* format, even if not all were fully structured. Items were prioritized using MoSCoW technique, though little change was made to the existing priority structure due to time constraints. To estimate effort, the team used the **Planning Poker estimation technique** (*see Appedix E*) with **Fibonacci scale** [10] assigning story points from (1 to 13) to reflect complexity, risk, and repetition. Estimated development time was tracked in man-days, providing a practical reference for sprint capacity.
In parallel, the team agreed on a shared **Definition of Done**, which specified the minimum criteria for a story to be considered complete, including coding, basic testing and review.
The Table 3.3 illustrates the user stories planed for the first Sprint.

Table 3.3: Sprint 1 – User Stories

ID	**User Story**	**Story Points**	**Priority**
US1	As a visitor, I want to see a clean and responsive homepage with a navigation menu, banner, and featured categories so that I can quickly understand what the site offers.	5	Must
US2	As a new customer, I want to create an account using my email and password so that I can save my profile and access order tracking.	3	Must
US3	As a registered user, I want to log in securely using my credentials so that I can manage my account and start shopping.	3	Must
US4	As a user, I want to browse a list of men's clothing items by category so that I can quickly find what I'm looking for.	5	Must
US6	As a site administrator, I want to upload product images through a secure admin interface so that new items can be visually showcased.	3	Should
US7	As a frontend developer, I want to build the UI using a mobile-first responsive design so that users have a seamless experience on any device.	5	Should
US8	As a developer, I want to organize the project using Git and create a clear branch structure so that the team can collaborate without conflicts.	2	Could

Sprint Execution

The sprint lasted two weeks and was managed without using formal tools like Jira. The team relied on shared understanding supported by direct communication. Tasks were assigned to specific developers, and stories were grouped under planned release versions. A board was used to illustrate to visualize progress and simulate a Scrum board (see Figure 3.4).

The sprint backlog (Table 3.3) were executed efficiently, and overall, defect fixing time remained low. This showed good code quality and clear task understanding among the team.

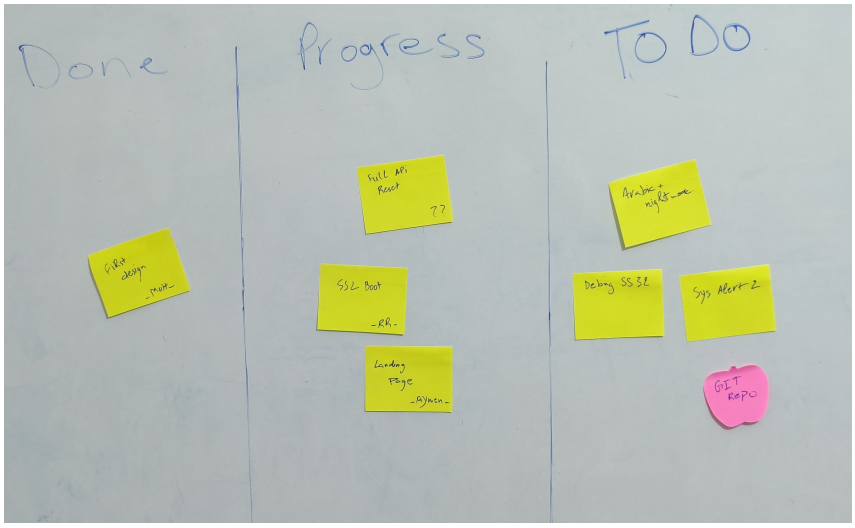


Figure 3.4: Scrum Board

Sprint Review

At the end of the sprint, a short review session was held with the Product Owner. The team demonstrated completed features, and feedback was collected for the next sprint according to the *SBOK* [41] (see Figure 3.6). Beta testing (with the client) phases were planned according to the backlog. Team members noted that:

- Development speed was good, with accurate estimations (Figure 3.5).
- Tasks involving UI dependencies required earlier clarification.
- Direct collaboration and in-person discussions worked better than expected.

B	C	D	E	F	G	H	I	J	K	L	M	N
Total Story Points In Scope	Total Man Days	Estimated Velocity	Total Story Points Delivered	Actual Velocity			Date	Man Days (Today)	Man Days (Cumulative)	Story Points (Est Cumulative)	Story Points (Actual Today)	Story Points (Actual Cumulative)
26	24	1,083333333	24,5	0,979591837		Week 1	25/01/2025	3	3	3,25	1	1
							26/01/2025	2	5	5,416666667	2	3
							27/01/2025	3	8	8,666666667	2	5
							28/01/2025	2	10	10,83333333	0,5	5,5
							29/01/2025	2	12	13	5	10,5
						Week 2	30/01/2025	4	16	17,33333333	2	12,5
							31/01/2025	0	16	17,33333333	2	14,5
							01/02/2025	4	20	21,66666667	3	17,5
							02/02/2025	3	23	24,91666667	8	25,5
							03/02/2025	1	24	26	0	25,5

Figure 3.5: The Sprint Tracking Sheet

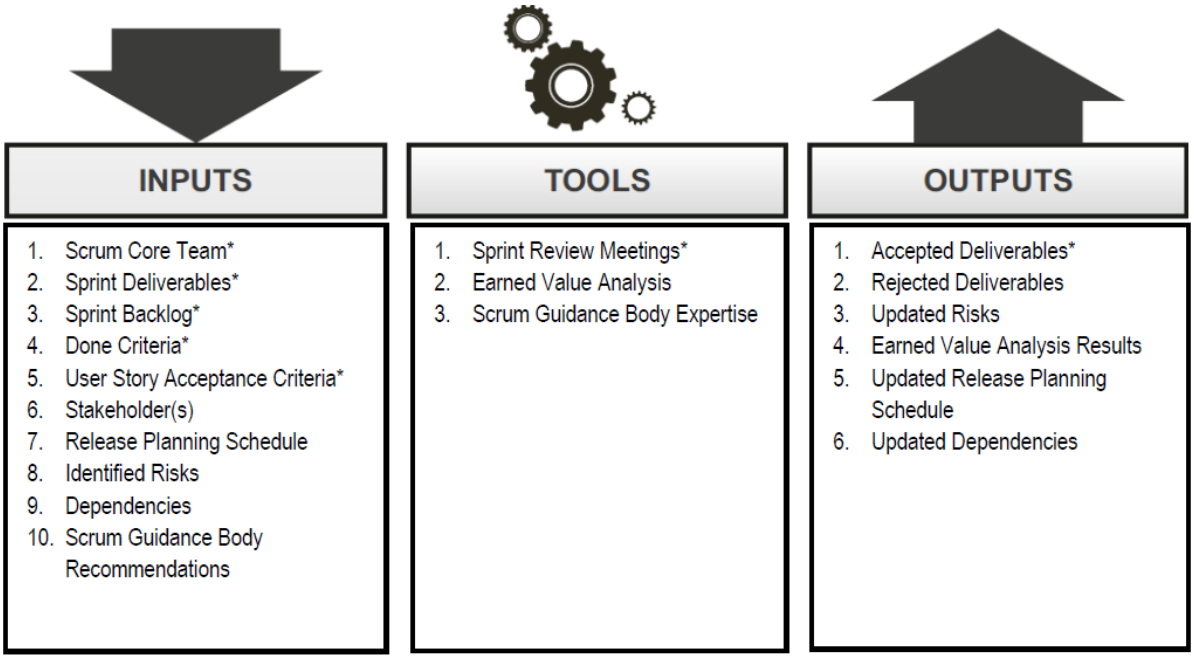


Figure 3.6: Sprint Review Process According to the SBOK

3.3.3 Step 3 : Continuous Improvement and Sustainability Integration

After Sprint 1, the team held a **retrospective session** to reflect on performance, identify issues, and plan adjustments. The goal was to reinforce continuous improvement while setting the stage for integrating lightweight sustainability practices in future sprints.

Using the tracking sheet (Figure 3.5) and the corresponding burndown chart (Figure 3.7), the team compared estimated and actual progress. Although the sprint started with minor delays, it ended with 25.5 out of 26 story points delivered, a sign of growing coordination and stable velocity.

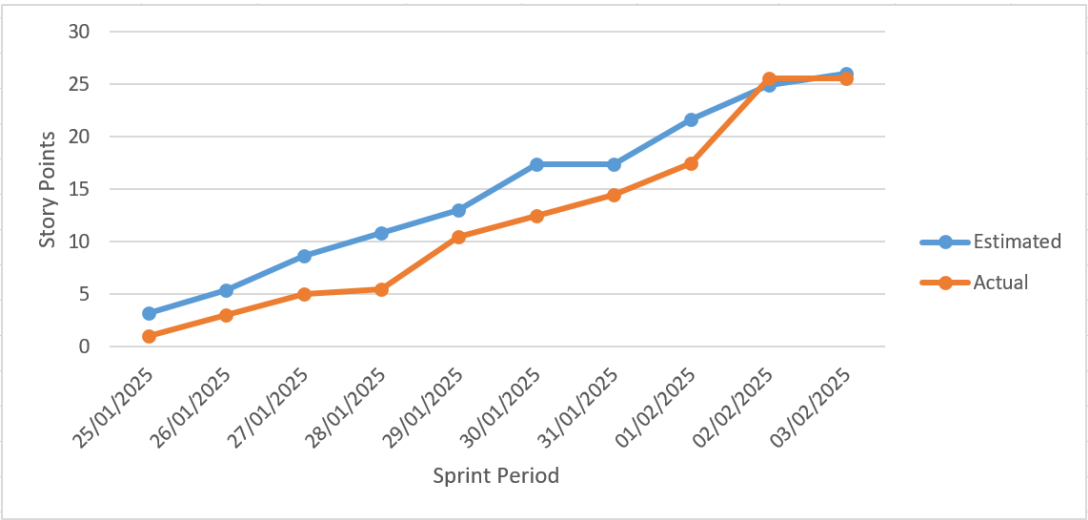


Figure 3.7: The Burndown Chart for Sprint 1

Based on Sprint 1's lessons, adjustments were made to how tasks would be split in Sprint

2. For example, better balancing of man-days across team members and clearer story definitions were noted as areas of focus. These refinements were directly influenced by the visual data, which showed where the team was most efficient and where blockers had occurred.

The team members automatically began considering ways to improve the sprint structure to reduce unnecessary effort and ensure smoother transitions between phases. These ideas: like refining testing cycles and reusing components, will be gradually incorporated as the project advances.

Sprint 1 laid the foundation not only for product delivery but also for process maturity. The improvements discussed in the retrospective, backed by clear visual feedback, will guide the setup and execution of the next sprints, maintaining the agile cycle of continuous learning and adaptation.

3.4 Conclusion

The first application of Scrum within this project confirmed its relevance to both the technical and operational realities of the team. Sprint 1 provided a working structure that improved collaboration, gave visibility to progress, and helped the team stay aligned with business priorities.The review and retrospective sessions showed tangible outcomes in terms of task completion and estimation accuracy, offering data that will guide the next sprints. More importantly, the team began to engage more deeply with process improvement as adjusting workload distribution, refining requirements, and setting up better coordination practices.

This chapter established the operational rhythm of the project and demonstrated how Scrum can serve as a flexible foundation to support not only delivery but also long-term improvement and sustainability goals.

Chapter 4

Evaluation and Predictive Impact of Sustainable Scrum Adoption

This chapter presents a quantitative exploration of how the adoption of Scrum and sustainable software engineering practices can influence key project performance indicators. Due to limited access to real-world operational data, a synthetic dataset was generated to simulate development scenarios with and without these practices.
The chapter begins by describing the data collection, then outlining the machine learning approach used to build predictive models, and how we can select the best ml model for our dataset. The objective is to assess how variables such as team velocity, rework hours, technical debt, and customer satisfaction respond to methodological changes.
By analyzing the predicted outcomes, we aim to validate the expected benefits of integrating Scrum and sustainability into the software development process. This serves both as a proof-of-concept and a foundation for future data-driven evaluations.

4.1 Dataset Overview

Since the simulation is based on a theoretical model, a synthetic dataset was constructed to reflect a realistic set of project scenarios. Each row in the dataset corresponds to a completed project, with various attributes collected to analyze productivity, quality, and sustainability-related outcomes.

4.1.1 Data Creation

The dataset used in this thesis was generated using the MOSTLY AI synthetic data platform. It was built from a small real-world dataset and project-specific constraints, inspired by observations during the e-commerce platform case study.

1. Base Data: A limited dataset of 6 rows was extracted based on patterns seen in real software projects, including key attributes like methodology, team size, delivery delay, and customer satisfaction.
2. Constraint Definition: Domain-specific rules and expected value ranges (e.g., velocity in Scrum vs. Waterfall) were defined to guide the generator.
3. Generation Process: MOSTLY AI's generator was trained on the base data, producing a larger synthetic dataset that reflects realistic project dynamics while ensuring data privacy.

This approach allowed us to model and analyze development outcomes, while maintaining confidentiality and consistency with real-world trends.

4.1.2 Dataset Description

Table 4.1 below is a description of each variable used in the dataset, and you can check **Appendix D** for a data sample.

Table 4.1: Project Parameters Description

Parameter	**Description**	**Data Type**	**Categories / Min–Max Values**
methodology	Development process methodology	Categorical	Scrum, Waterfall
start date & end date	Timeline boundaries of the project. Used to calculate duration.	Date	1995 – 2023
team size	Number of team members assigned to the project.	Integer	4 – 15
team velocity	A measure of delivery speed (e.g., story points completed per sprint).	Float	∼2 – 50
technical debt	Estimation of long-term maintainability problems (scale or score).	Float	0 – 146
sustainability practices adopted	Indicates whether sustainability principles (e.g., automation, green hosting) were applied. Values: Yes or No.	Categorical	Yes, No
defect reported	Number of bugs or issues reported during testing or production.	Integer	1 – 19
rework hours	Total time spent fixing bugs or redoing parts of the system.	Float	0 – 78
customer satisfaction	Score reflecting client satisfaction post-delivery (scale 1–10 or percentage).	Float	4 – 10
budget	Total budget allocated for the project.	Integer	3,168 – 464,224 (USD)
delivery delay days	Number of days the project was delayed beyond the planned deadline. Negative means early delivery and positive means delay.	Float	-34 to +14

These parameters were selected to reflect both project **performance** and **sustainability impact**, allowing a comparative analysis of different development approaches. By structuring the data in this way, the analysis in the following sections can examine the influence of methodology and sustainability integration on key indicators such as quality, cost, and delivery time.

4.2 Prediction Objectives

The goal is to evaluate how adopting Scrum and sustainability-oriented practices may influence key project outcomes. To do this, four main metrics were selected as prediction targets, each aligned with one of the pillars of Sustainable Software Engineering:
Table 4.2 summurizes each prediction target and SSE pilars.

Table 4.2: Prediction Metrics and Sustainability Dimensions

Metric	**Description**	**Sustainability Pillar**
Team Velocity	Measures how much work the team completes over time. Reflects productivity and delivery rhythm.	Social – team well-being and flow
Technical Debt	Represents accumulated shortcuts in code quality or maintainability.	Technical – long-term maintainability
Rework Hours	Time spent fixing bugs or repeating work due to earlier errors. Indicates process inefficiency.	Environmental – resource and energy efficiency
Customer Satisfaction	Measures the perceived value and usability of the delivered product.	Economic – value delivery and stakeholder satisfaction

These indicators provide a balanced view of project performance while linking back to SSE concerns such as resource efficiency, team sustainability, and user-centered development. By predicting these values based on methodology and sustainability actions, we can estimate the practical benefits of transitioning to Scrum and integrating eco-responsible practices.

4.3 Data Processing

To ensure reliable and consistent results, the dataset was preprocessed using the Orange Data Mining platform. This step was essential to prepare the data for machine learning by addressing common quality and formatting issues automatically. As shown in Figure 4.3, Orange provides several preprocessing options that are applied in the background or selected manually when needed.

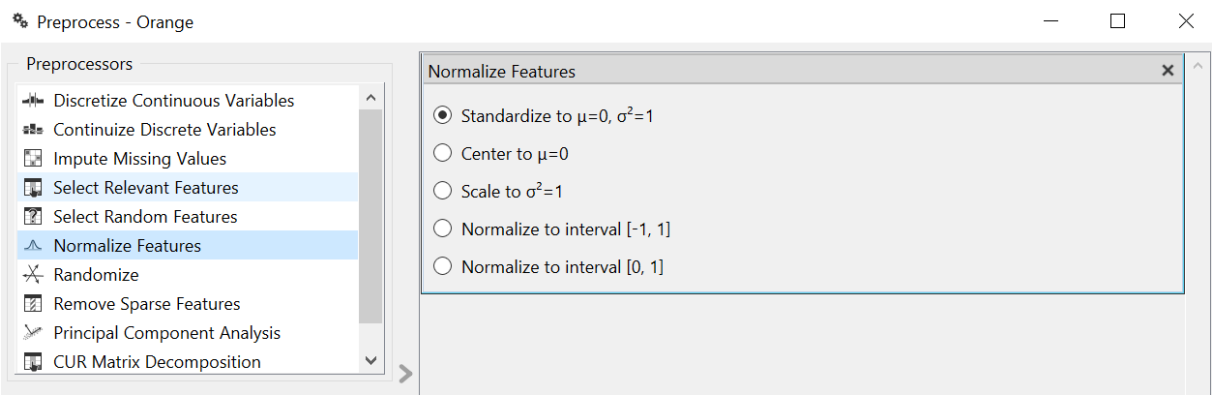


Figure 4.1: Automatic data preprocessing in Orange

In our case, the platform automatically managed missing values by either imputing them or removing incomplete rows, depending on the severity of the gaps. It also filtered out low-variance or redundant features to retain only those most relevant for prediction. As shown in Figure 4.1, numerical features were normalized to follow a standard distribution, with a mean of 0 and a standard deviation of 1, ensuring fair treatment of variables during model training. Additionally, Orange excluded irrelevant or inconsistent columns based on internal rules, further improving data quality and uniformity.

This automated data cleaning ensured that all predictive models were trained on consistent and high-quality inputs, reducing the risk of overfitting, bias, or poor generalization. The process contributed to improving model accuracy and fairness by minimizing the influence of noise or outliers in the data.

4.4 Model Selection and Evaluation

To predict the outcomes of Scrum adoption and sustainability practices, a supervised machine learning approach was applied using the Orange ML platform. Four different regression models were tested: **AdaBoost, k-Nearest Neighbors (kNN), Linear Regression, and Support Vector Machine (SVM)**. Each model was trained and evaluated using the same dataset, and their performance was compared based on common error metrics [13, 38].

Model	MSE	RMSE	MAE	MAPE	R2
AdaBoost	1.484	1.218	0.572	0.097	0.777
kNN	4.768	2.183	1.658	0.286	0.284
Linear Regression	5.836	2.416	2.086	0.353	0.124
SVM	6.333	2.517	1.903	0.363	0.049

Figure 4.2: Comparison of Regression Models

The workflow in Figure 4.3 shows how Orange was used to configure, train, and test the models. The dataset was connected to multiple learners, followed by a "Test and Score" block to compare performance. The best-performing model was then connected to a prediction widget to simulate real project scenarios.
Among the tested models, AdaBoost clearly outperformed the others (see Figure 4.2), achieving the **lowest** Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and Mean Absolute Error (MAE), along with the **highest** R^2 score of 0.777. This indicates that AdaBoost explained approximately 78% of the variance in the outcome variables, making it a reliable choice for prediction in this context [13].

Based on these results, **AdaBoost was selected as the final model** to simulate the impact of methodological choices and sustainability practices on key project outcomes, including team velocity, technical debt, customer satisfaction and rework hours.

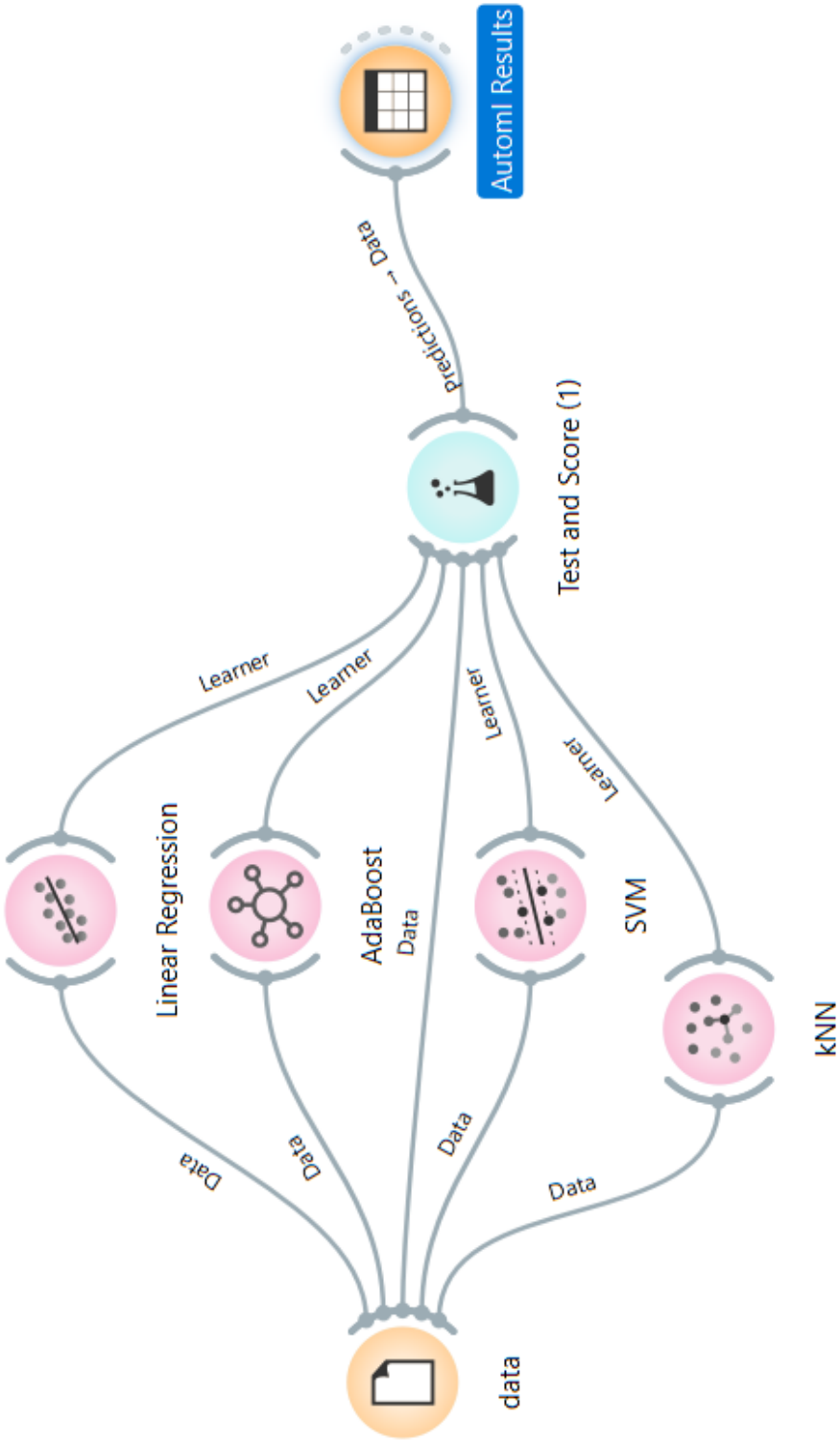


Figure 4.3: Models Testing Workflow Using Orange DM

4.5 Prediction Results Analysis

To evaluate the impact of integrating Scrum and sustainability practices, two simulated project instances were compared using AdaBoost model, see Figure 4.4. Both instances share similar parameters, including team size, project scope, and duration. The only differences lie in the development methodology (Waterfall vs. Scrum) and the adoption of sustainability practices (Yes vs. No). So the **before** instance include **(Waterfall,No)** pair, and the **after** instance includes **(Scrum,Yes)** pair.

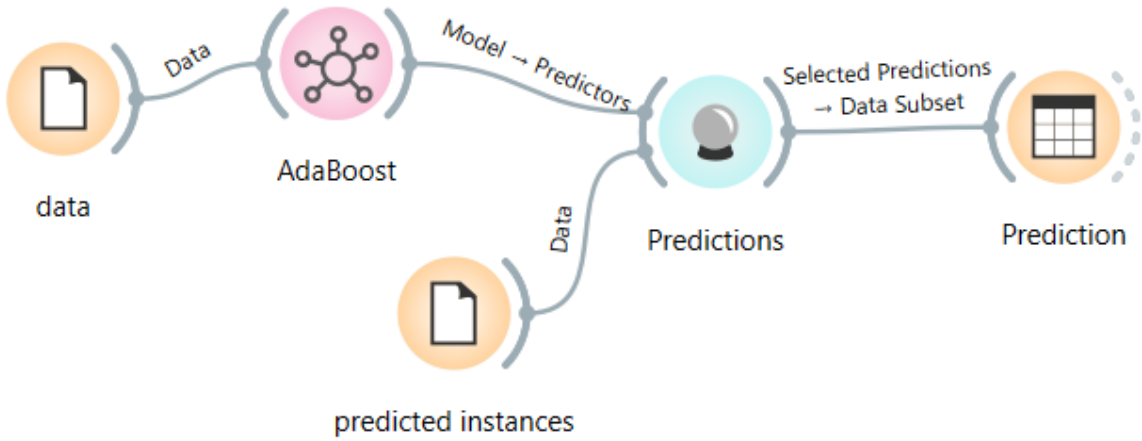


Figure 4.4: AdaBoost Workflow

This controlled comparison helps isolate the effects of these variables on key outcome metrics. We compared the project's performance before and after their integration. The table below presents the evolution of four key indicators mapped to sustainability pillars.

Table 4.3: Key Performance Metrics: Before and After Scrum + SSE

Metric	Before	After	Scale	Change (%)
Customer Satisfaction	7.0	9.0	0–10	+28.6%
Technical Debt	81.0	61.3	0–146	−24.3%
Team Velocity	26.0	36.0	0–50	+38.5%
Rework Hours	42.0	37.0	0–78	−11.9%

Interpretation:

- **Customer Satisfaction** increased by nearly 29%, showing a positive impact on perceived product value and client alignment.
- **Technical Debt** dropped by over 24%, reflecting cleaner code, fewer long-term issues, and improved development discipline.
- **Team Velocity** saw a 38.5% improvement, indicating higher productivity and better task planning under the new workflow.
- **Rework Hours** were reduced by nearly 12%, showing improved first-time quality and better early-stage feedback handling.

To better illustrate the impact of the applied methods, the following Figure 4.5 presents the percentage change observed across the four selected outcome metrics. Improvements

in customer satisfaction and team velocity demonstrate a more collaborative and efficient working environment, while the reductions in technical debt and rework hours reflect enhanced code quality and reduced waste. These results confirm the positive contribution of Scrum and sustainability-oriented practices in strengthening both project performance and long-term maintainability.

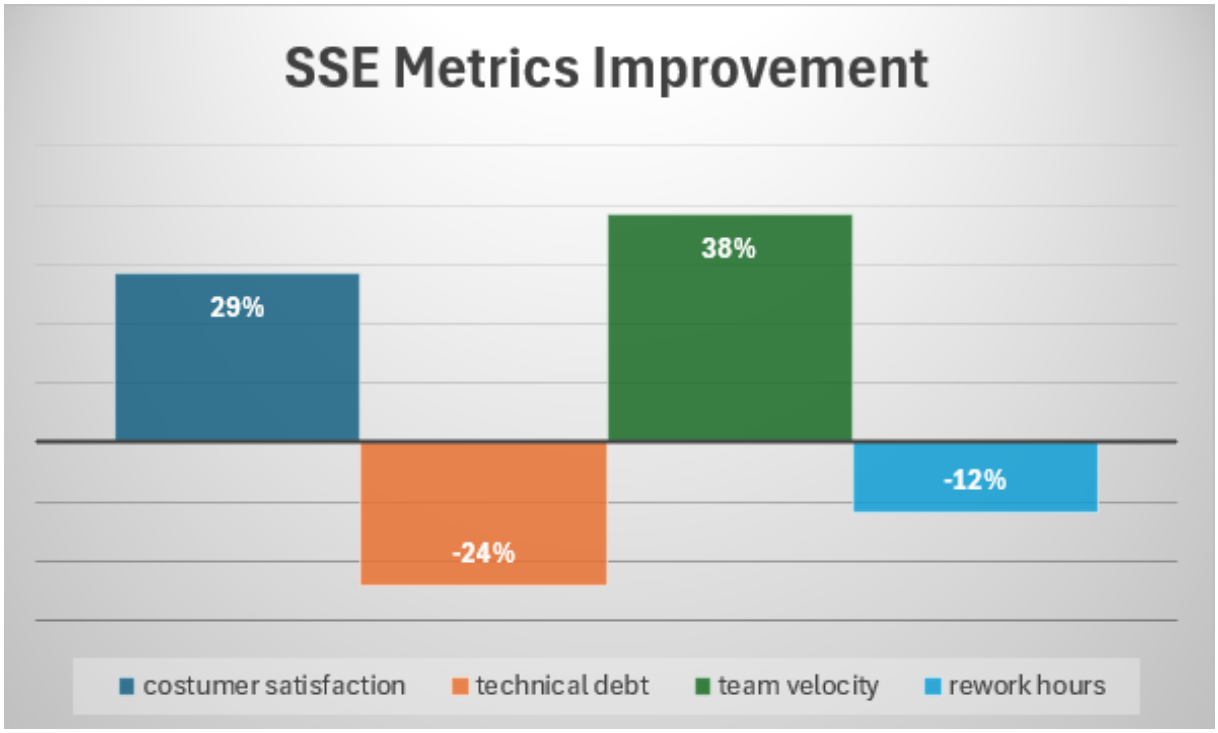


Figure 4.5: Improvement of SSE Metrics After Scrum & Sustainability Integration

Overall, the integration of Scrum and sustainability principles resulted in measurable improvements across technical, economic, and social performance indicators.

4.6 Limitations

Several limitations were encountered during this study, which should be considered when interpreting the results:

- **Lack of Real Project Data and Use of Synthetic Dataset:** Due to confidentiality constraints and gaps in documentation, no operational or performance data could be obtained from the internship host company. As a result, a synthetic dataset was created to simulate realistic project scenarios. While this dataset was carefully designed to reflect diverse development conditions, it remains artificial and may not fully capture the complexities and nuances of real-world software environments.

- **Unsuccessful BPMN Simulation:** An attempt was made to simulate the process using BPMN modeling tools, but the effort did not yield usable results. The primary obstacle was the absence of detailed time logs or measurable transition data across phases.

- **Restricted Access to Internal Documents:** Documentation related to code quality, deployment metrics, or resource usage was either unavailable or outdated, limiting the possibility of conducting data-driven analysis based on actual project flow.

- **Simplified Assumptions:** Several assumptions were made to complete the simulation, including fixed velocity for Scrum teams and uniform effort distribution across roles. These simplifications, while necessary for modeling, reduce the precision of predictions.

These constraints limit the generalizability of the findings. However, the exercise remains valuable for illustrating how sustainability indicators can be modeled and analyzed when integrated with agile practices.

4.7 Conclusion

The results presented in this chapter demonstrate clear improvements across key metrics when Scrum and sustainability-oriented practices are applied. Team velocity increased, rework hours and technical debt decreased, and customer satisfaction improved significantly. These gains reflect the advantages of iterative development, early feedback integration, and better resource alignment.

Despite the simplified nature of the dataset, the simulation provides strong initial evidence supporting the use of modern, sustainable methodologies. It also shows that even synthetic modeling can offer meaningful insights when real-world data is scarce.
This experimentation confirms the potential of predictive analytics in software engineering and opens new opportunities for integrating sustainability metrics into agile project evaluation.

General Conclusion

This thesis explored the integration of Sustainable Software Engineering and engineering methodologies within a simulated e-commerce development project, aiming to improve both technical and sustainability outcomes in software processes. Through a combination of bibliographic study, process diagnosis, and implementation, the study provided a structured response to common inefficiencies encountered in traditional development models.

The bibliographic review established the theoretical foundation of SSE, identifying its key pillars: environmental, economic, social, and technical, and mapping relevant tools like Scrum, Kanban, Value Stream Mapping, and LCA to the SDLC. In the case study at Icosnet, the existing Waterfall model was analyzed using diagnostic tools such as SWOT, Ishikawa, and 5 Whys. These tools revealed process rigidity, lack of automation, delayed client feedback, and missing sustainability indicators as major weaknesses.

To address these gaps, a new SDLC structure was proposed using Scrum as the core delivery framework. The simulation covered sprint planning, execution, and retrospective cycles, with adaptations to reflect internal team realities. The iterative model improved client collaboration, increased flexibility, and opened the door for sustainability actions such as improved testing discipline and better resource allocation.

Chapter 4 extended the contribution by evaluating the simulated project's performance through a predictive machine learning model based on synthetic data. Using Adaboost, key outcome variables: team velocity, technical debt, rework hours, and customer satisfaction, were analyzed before and after the integration of Scrum and SSE principles. Results showed measurable improvements in all metrics, validating the potential impact of such practices even in a simulated setting.

Due to limited access to real company data and project KPIs, synthetic data was generated to model the scenario. Tools like BPMN simulation were also explored, but could not be used effectively due to insufficient process data. Nevertheless, the experiment confirmed the potential of combining Agile thinking with sustainability indicators to create more adaptive, efficient, and responsible development workflows.

Beyond the simulation, this study suggests that sustainable software engineering is not limited to environmental factors, it is a holistic mindset that supports continuous improvement, team alignment, and value delivery. Future work could apply these findings to real-world projects, expand the use of sustainability metrics, and refine the prediction models using live project datasets.

This thesis contributes to the growing body of knowledge on sustainable digital practices in Algeria, highlighting that local companies like Icosnet can benefit from lightweight, structured improvements that bridge performance and sustainability.

Appendices

Appendix A : Business Model Canvas BMC

The Business Model Canvas (BMC) provides a structured and comprehensive view of the strategic positioning of the e-commerce platform within the Algerian market. Each component of the canvas reflects a deliberate choice that aligns with the project's objectives and ICOSNET's digital strategy :

Table 1.4: Business Model Canvas

Key Component	**Details**
Customer Segments	Primary Market: Algerian men seeking high-quality, stylish clothing online. Looking for premium but affordable brands.
Value Proposition	For Consumers: High-quality, stylish men's clothing with fast, reliable delivery and flexible payment options.
Channels	Digital Storefront: A web and mobile-optimized e-commerce site. Social Media & Marketing: Facebook, Instagram, TikTok campaigns, influencer collaborations.
Customer Relationships	Customer Support: 24/7 chatbot, email, and phone assistance. Personalized Shopping Experience: loyalty programs, exclusive discounts.
Revenue Streams	Direct E-Commerce Sales: Selling products online. Limited Edition Drops: Exclusive, time-limited collections to create urgency.
Key Resources	Technology: Cloud-based e-commerce system. Brand & Inventory: High-quality men's fashion products, stock management. Human Resources: Development team, digital marketing specialists, customer support. Logistics Partnerships: Local integrations for delivery efficiency.
Key Activities	E-Commerce Platform Management: Ensuring site stability, updates, and security. Order Fulfillment & Customer Service: Managing deliveries, returns, and customer engagement.
Key Partnerships	Fashion Suppliers: Sourcing and manufacturing high-quality products. Financial Institutions: Payment processing partnerships. Delivery Services: Agreements with third-party logistics providers.
Cost Structure	Technology Infrastructure: Server costs, API integrations, payment gateway fees. Operational Costs: Salaries, inventory management.

Appendix B : Scrum Framework

As part of the project documentation, this appendix includes a comprehensive Scrum cheat sheet [1]. The goal of this reference is to provide a clear and accessible summary of the Scrum framework as applied throughout the development of the e-commerce platform. This cheat sheet includes the main components of Scrum, such as:

- The **three core roles**: Product Owner, Scrum Master, and Development Team.
- The **five Scrum events**: Sprint, Sprint Planning, Daily Scrum, Sprint Review, and Sprint Retrospective.
- The **three artifacts**: Product Backlog, Sprint Backlog, and Increment.
- Estimation techniques such as **Planning Poker**, **Story Points**, and **T-shirt sizing**.
- Workflow tools like **Burn Down Chart**, **Burn Up Chart**, and **Definition of Done**.
- Key principles such as **transparency**, **inspection**, and **adaptation**.

This cheat sheet served as a practical guide for the team throughout the simulation of the Scrum process. It helped align all members on common terminology, clarified processes, and supported decision-making during Sprint events. For a team operating in an academic context, especially one composed of junior developers and interns, this document proved essential in reinforcing Agile discipline and consistency.

Including it in the appendix ensures that any reader or evaluator can easily understand the Scrum structure that was followed, and how it guided the management and execution of the project from Sprint 0 to delivery.

SCRUM CHEAT SHEET

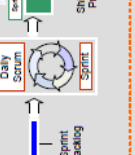
Roles

Scrum Team
- Team is cross-functional and consists of 5-9 people
- There are no set project roles within the team
- Team defines tasks and assignments
- Team is self-organizing and self-managing
- Maintains the Sprint Backlog
- Conducts the Sprint Review

Product Owner (PO)
- Accountable for product success
- Defines all product features
- Responsible for prioritizing product features
- Maintains the Product Backlog
- Insures team working on highest valued features

Scrum Master (SM)
- Holds daily 15 minute team meeting (Daily Scrum)
- Removes obstacles
- Shields the team from external interference
- Maintains the Sprint Burndown Chart
- Conducts Sprint Retrospective at the end of a Sprint
- Is a facilitator not a manager

Process



Tools

Task Board
- White Board containing teams Sprint goals, backlog items, tasks, tasks in progress, "DONE" items and the daily Sprint Burndown chart.
- Scrum meeting best held around task board
- Visible to everyone

Artifacts

Product Backlog - (PB)
- List of all desired product features
- List can contain bugs, and non-functional items
- Product Owner responsible for prioritizing
- Items can be added by anyone at anytime
- Each item should have a business value assigned
- Maintained by the Product Owner

Sprint Backlog – (SB)
- To-do list (also known as Backlog item) for the Sprint
- Created by the Scrum Team
- Product Owner has defined as highest priority

Burndown Chart – (BC)
- Chart showing how much work remaining in a Sprint
- Calculated in hours remaining
- Maintained by the Scrum Master daily

Release Backlog – (RB)
- Same as the Product Backlog. May involve one or more sprints dependent on determined Release date

"DONE"= Potentially Shippable!

FAQ
- *Who decides when a Release happens?* At the end of any given Sprint the PO can initiate a Release.
- *Who is responsible for managing the teams?* The teams are responsible for managing themselves.
- *What is the length of a task?* Tasks should take no longer than 16 hours. If longer then the task should be broken down further.
- *Who manages obstacles?* Primary responsibility is on the Scrum Master. However, teams must learn to resolve their own issues. If not able then escalated to SM.
- *What are two of the biggest challenges in Scrum?* Teams not self-managing, Scrum Master managing not leading.

Meetings

Sprint Planning – Day 1 / First Half
- Product backlog prepared prior to meeting
- First half – Team selects items committing to complete
- Additional discussion of PB occurs during actual Sprint

Sprint Planning – Day 1 / Second Half
- Occurs after first half done – PO available for questions
- Team solely responsible for deciding how to build
- Tasks created / assigned – Sprint Backlog produced

Daily Scrum
- Held every day during a Sprint
- Lasts 15 minutes
- Team members report to each other not Scrum Master
- Asks 3 questions during meeting
- *"What have you done since last daily scrum?"*
- *"What will you do before the next daily scrum?"*
- *"What obstacles are impeding your work?"*
- Opportunity for team members to synchronize their work

Sprint Review
- Team presents "done" code to PO and stakeholders
- Functionality not "done" is not shown
- Feedback generated - PB maybe reprioritized
- Scrum Master sets next Sprint Review

Sprint Retrospective
- Attendees – SM and Team. PO is optional
- Questions – What went well and what can be improved?
- SM helps team in discovery – not provide answers

Visibility + Flexibility = Scrum

Glossary of Terms
- **Time Box** - A period of time to finish a task. The end date is set and can not be changed
- **Chickens** – People that are not committed to the project and are not accountable for deliverables
- **Pigs** – People who are accountable for the project's success
- **Single Wringable Neck** – This is the Product Owner!

Estimating

User Stories
- A very high level definition of what the customer wants the system to do.
- Each story is captured as a separate item on the Product Backlog
- User stories are NOT dependent on other stories
- **Story Template**:
- "As a <User> I want <function> So that <desired result>
- **Story Example**:
- As a user, I want to print a recipe so that I can cook it.

Story Points
- A simple way to initially estimate level of effort expected to develop
- Story points are a relative measure of feature difficulty
- Usually scored on a scale of 1-10. 1=very easy through 10=very difficult
- **Example**:
- "Send to a Friend" Story Points = 2
- "Shopping Cart" Story Points = 9

Business Value
- Each User Story in the Product Backlog should have a corresponding business value assigned.
- Typically assign (L,M,H) Low, Medium, High
- PO prioritizes Backlog items by highest value

Estimate Team Capacity
- Capacity = # Teammates (Productive Hrs x Sprint Days)
- Example – Team size is 4, Productive Hrs are 5, Sprint length is 30 days.
- Capacity = 4 (5 x30) = 600 hours
- **NOTE**: Account for vacation time during the Sprint!

Velocity
- The rate at which team converts items to "DONE" in a single Sprint – Usually calculated in Story Points.

Figure B.6: Scrum Cheat Sheet

Appendix C : Roles organization in Scrum

This appendix presents the role organization within the Scrum framework, based on the Scrum Body of Knowledge (SBOK) [41] and adapted to the structure of this project. It provides an overview of how responsibilities are distributed among different team members, which is essential for understanding the collaborative and iterative nature of Scrum. Scrum roles are categorized into two main types:

- **Core Roles**: These are directly accountable for delivering the product.
  - **Product Owner (PO)** – Represents the customer, defines the Product Backlog, and ensures the team delivers value aligned with business priorities.
  - **Scrum Master (SM)** – Facilitates the Scrum process, ensures adherence to Agile principles, and removes any obstacles that may affect progress.
  - **Development Team** – A cross-functional, self-organizing group responsible for delivering potentially shippable increments at the end of each Sprint.
- **Non-Core Roles**: These may be involved in the project but are not part of the Scrum Team.
  - **Stakeholders** – Include users, customers, or executives who provide feedback and contribute to defining high-level requirements.
  - **Vendors** – External parties who provide products or services required during development.

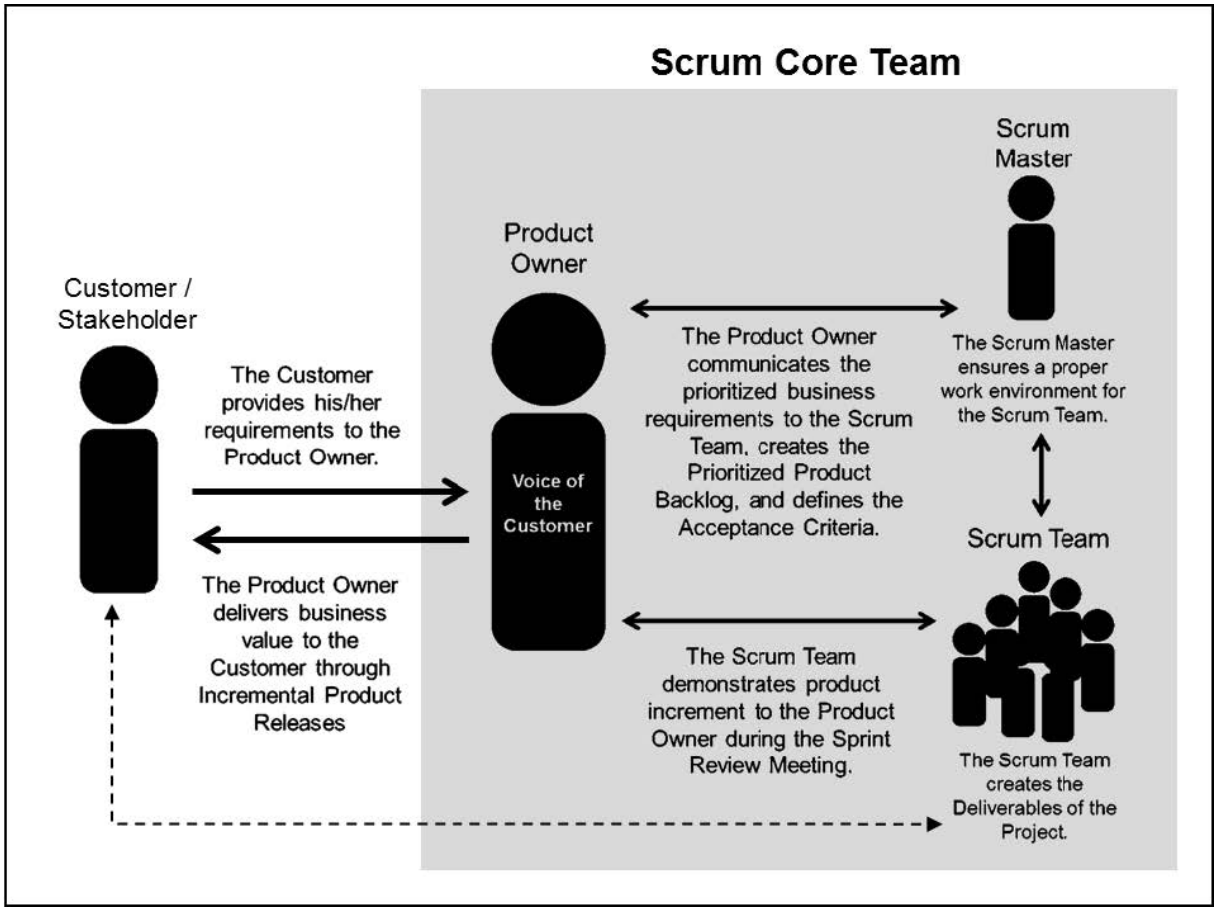


Figure C.7: Scrum Roles

Appendix D : Dataset sample

To support the evaluation of the proposed methodology, a synthetic dataset was constructed, reflecting realistic scenarios inspired by the simulated e-commerce case study. The dataset contains project-level information across various development contexts, comparing traditional (Waterfall) and Agile (Scrum) methodologies, with or without the integration of sustainability practices. Each entry simulates a full software development project with defined input variables (e.g., team size, methodology, sustainability actions) and outcome metrics (e.g., team velocity, customer satisfaction, technical debt).
The sample shown below illustrates a subset of this dataset. It includes both categorical and numerical variables, such as project methodology type, defect rates, rework hours, and delivery delays. These entries were designed to reflect variability in project conditions while controlling for factors like domain, duration, and budget scale. This allowed the machine learning models (explained in Chapter 4) to detect patterns in project outcomes, assess the impact of Scrum and SSE, and predict improvements with reasonable generalization.

	project_id	methodology	start_date	end_date	team_size	team_velocity	technical_debt	able_practices_a	defects_reported	rework_hours	stomer_satisfacti	budget_usd	elivery_delay_day
113	36	Scrum	1997-11-27	2004-06-03	4	25	4	No	19	4	9	102526	-34
122	40	Waterfall	2010-08-10	2011-09-23	14	32	146	No	11	20	10	80149	6
174	43	Scrum	2015-07-21	2018-01-20	15	35	51	No	2	48	10	17267	14
82	52	Waterfall	2007-09-12	1995-04-20	10	33	9	No	12	35	4	40897	14
1	56	Waterfall	2001-01-25	2007-12-11	4	50	61	Yes	18	40	10	9526	4
209	102	Waterfall	1999-02-09	2023-01-31	14	50	146	No	15	11	9	137508	14
138	103	Waterfall	2011-04-22	2015-12-17	14	13	11	No	13	20	10	17435	-2
194	103	Scrum	2017-03-26	2004-01-01	4	2	9	No	19	34	10	69804	14
62	117	Waterfall	1994-05-20	2007-04-26	4	50	119	No	14	77	4	41237	3
150	135	Scrum	2013-07-11	2018-09-15	4	36.2334	61.322	Yes	16	21.7628	9.87878	219609	-6.76098
107	136	Scrum	2005-03-13	2002-04-12	5	36.2334	61.322	Yes	16	21.7628	9.87878	10561	-6.76098
4	157	Scrum	2003-12-08	2001-09-01	4	4	24	No	14	73	4	13490	-4
247	164	Scrum	1999-07-19	2003-07-19	15	41	146	No	13	32	4	13753	12
137	165	Scrum	1994-11-27	2010-05-23	15	36.2334	61.322	Yes	7	21.7628	9.87878	25190	-6.76098
243	187	Waterfall	2014-09-19	2009-07-17	15	50	144	No	4	3	9	73836	-30
17	194	Scrum	2020-07-04	2009-12-23	4	50	99	No	19	42	9	48059	4
175	215	Scrum	2012-05-25	1999-10-04	10	8	146	No	13	74	7	60290	-7
155	234	Scrum	1996-03-24	2006-11-19	14	32	49	No	2	66	4	387016	-12
226	276	Waterfall	2008-03-03	2018-08-26	5	21	146	Yes	16	45	10	32621	14
15	309	Waterfall	2000-07-19	2015-09-30	14	9	118	No	12	7	10	464224	-1
123	312	Scrum	2016-08-01	2018-06-26	10	36.2334	61.322	Yes	4	21.7628	9.87878	217235	-6.76098
134	416	Waterfall	2005-01-05	1998-12-17	9	47	140	Yes	13	33	4	336610	2
234	499	Waterfall	2015-10-20	2004-04-02	14	28	49	No	16	37	10	11068	-16
195	515	Waterfall	2013-07-11	2001-06-26	15	50	146	No	18	1	4	20735	-34
68	695	Scrum	2011-01-25	1999-11-28	15	50	90	No	9	37	10	268470	-14
145	777	Scrum	2006-09-17	2020-06-22	4	2	146	No	15	58	4	39503	0
121	812	Scrum	1997-06-18	2021-01-31	15	36.2334	61.322	Yes	3	21.7628	9.87878	34743	-6.76098
146	1001	Waterfall	2004-11-04	2012-08-28	4	20	8	Yes	9	73	6	43045	-32
153	1010	Scrum	2000-12-08	2012-05-23	4	20	146	No	15	3	10	455511	-14
22	1022	Waterfall	2010-05-01	2019-12-24	10	50	62	No	10	19	5	502115	0
27	1034	Scrum	1998-10-07	2004-01-14	15	50	146	No	10	41	9	184984	14
207	1340	Scrum	1995-06-16	2003-11-21	15	36.2334	61.322	Yes	18	21.7628	9.87878	217268	-6.76098
74	23	Waterfall	2010-04-24	2022-09-15	5	3	96	No	5	78	4	87777	4
187	104	Waterfall	2003-10-14	2010-06-23	4	42	146	No	10	20	10	60723	-5

Figure D.8: Dataset Sample

Appendix E : Planning Poker Estimation Technique

What is it: Planning Poker, sometimes called Estimation Poker [41], is a team-based method used in Agile frameworks like Scrum to estimate the relative size or complexity of user stories. Inspired by the Wideband Delphi technique, it helps teams reach consensus while avoiding the influence of strong opinions or early bias. The technique encourages discussion, improves shared understanding, and supports more accurate planning decisions.

How it works: Each team member is given a set of numbered cards, typically using a scale such as 1, 2, 3, 5, 8, 13, and so on. These numbers do not represent exact time in hours but rather relative effort or complexity. Once the Product Owner explains the item to be estimated (usually a user story or a task), team members may ask clarifying questions to better understand the work involved. Then, without discussing estimates openly, each participant selects a card privately based on their personal judgment.
All cards are revealed simultaneously. If all team members choose the same number, the estimation is considered accepted. However, when there are significant differences, the team discusses why certain members chose higher or lower values. This discussion often uncovers hidden assumptions or overlooked aspects of the task. After the discussion, the team votes again. This cycle can be repeated until the group reaches agreement or selects a value that reflects a reasonable average.

Why it is useful: By combining independent thinking and group discussion, Planning Poker helps prevent groupthink and allows everyone's opinion to be heard. It also supports more realistic estimates by encouraging conversation around the uncertainty and risks of each item. The method is especially effective in small agile teams where collaboration, communication, and collective ownership are valued. It is a simple yet powerful way to improve estimation quality and strengthen team dynamics.

Bibliography

[1] Scribd - scrum cheat sheet. `https://fr.scribd.com/doc/200654496/Scrum-Cheat-Sheet`, 2025. Last Accessed: 2025-06-13.

[2] Muhammad Ovais Ahmad, Denis Dennehy, Kieran Conboy, and Markku Oivo. Kanban in software engineering: A systematic mapping study. *Journal of Systems and Software*, 137:96–113, 2018.

[3] Jide ET Akinsola, Afolakemi S Ogunbanwo, Olatunji J Okesola, Isaac J Odun-Ayo, Florence D Ayegbusi, and Ayodele A Adebiyi. Comparative analysis of software development life cycle models (sdlc). In *Intelligent Algorithms in Software Engineering: Proceedings of the 9th Computer Science On-line Conference 2020, Volume 1 9*, pages 310–322. Springer, 2020.

[4] Nauman Bin Ali, Kai Petersen, and Breno Bernard Nicolau De França. Evaluation of simulation-assisted value stream mapping for software product development: Two industrial cases. *Information and software technology*, 68:45–61, 2015.

[5] Nauman Bin Ali, Kai Petersen, and Kurt Schneider. Flow-assisted value stream mapping in the early phases of large-scale software development. *Journal of Systems and Software*, 111:213–227, 2016.

[6] J Almeida, ML Pellec, and J Bengtsson. Reusable coffee cups life cycle assessment and benchmark. *Edge: Joondalup, Australia*, 2018.

[7] Anders SG Andrae and Tomas Edler. On global electricity usage of communication technology: trends to 2030. *Challenges*, 6(1):117–157, 2015.

[8] Nala Freedrikson Arifin, Betty Purwandari, and Farisya Setiadi. Evaluation and recommendation for scrum implementation improvement with hybrid scrum maturity model: a case study of a new telco product. In *2020 International Conference on Informatics, Multimedia, Cyber and Information System (ICIMCIS)*, pages 178–183. IEEE, 2020.

[9] Lotfi Belkhir and Ahmed Elmeligi. Assessing ict global emissions footprint: Trends to 2040 & recommendations. *Journal of cleaner production*, 177:448–463, 2018.

[10] Pierre Bourque and R.E. Fairley. *Guide to the Software Engineering Body of Knowledge - SWEBOK V3.0*. 01 2014.

[11] Víctor Camañes, Rafael Tobajas, and Angel Fernandez. Methodology of eco-design and software development for sustainable product design. *Sustainability*, 16(7):2626, 2024.

[12] Mikael Octavinus Chan, Setiadi Yazid, et al. A novel framework for information security during the sdlc implementation stage: A systematic literature review. *Jurnal RESTI (Rekayasa Sistem dan Teknologi Informasi)*, 8(1):88–99, 2024.

[13] Davide Chicco, Matthijs J Warrens, and Giuseppe Jurman. The coefficient of determination r-squared is more informative than smape, mae, mape, mse and rmse in regression analysis evaluation. *Peerj computer science*, 7:e623, 2021.

[14] Tom Clancy. The standish group report. *Chaos report*, 1995.

[15] Erika Corona, Filippo Eros Pani, et al. A review of lean-kanban approaches in the software development. *WSEAS transactions on information science and applications*, 10(1):1–13, 2013.

[16] Christian Dawson and Ray Dawson. Software development process models: A technique for evaluation and decision-making. *Knowledge and Process Management*, 21(1):42–53, 2014.

[17] Monday Eze and Charles Okunbor. Analytical study of software development process model variants. *Global Journal of Engineering and Technology Advances*, 8(2):23–31, 2021.

[18] Matthias Galster. Life-cycle assessment in software engineering. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering*, 2010.

[19] Anne Gärtner, Daniel Leising, and Felix D Schönbrodt. Towards responsible research assessment: How to reward research quality. *PLoS biology*, 22(2):e3002553, 2024.

[20] Erol Gelenbe and Yves Caseau. The impact of information technology on energy consumption and carbon emissions. *ubiquity*, 2015(June):1–15, 2015.

[21] Ivan T Herrmann and Andreas Moltesen. Does it matter which life cycle assessment (lca) tool you choose?–a comparative assessment of simapro and gabi. *Journal of Cleaner Production*, 86:163–169, 2015.

[22] Amjad Hudaib, Raja Masadeh, Mais Haj Qasem, Abdullah Alzaqebah, et al. Requirements prioritization techniques comparison. *Modern Applied Science*, 12(2):62, 2018.

[23] IRENA. *Renewable Energy Statistics 2019*. The International Renewable Energy Agency, Abu Dhabi, 2019.

[24] Bong Keun Jeong, Soo-Yeon Ji, and Dong Hyun Jeong. Lean it with value stream mapping analysis: A case study in software development life cycle process. *Information Resources Management Journal (IRMJ)*, 35(1):1–18, 2022.

[25] Raisian Komeil, Yahaya Jamaiah, and Deraman Aziz. Current challenges and conceptual model of green and sustainable software engineering. 2016.

[26] Tatiana Kravchenko, Tatiana Bogdanova, and Timofey Shevgunov. Ranking requirements using moscow methodology in practice. In *Computer Science On-line Conference*, pages 188–199. Springer, 2022.

[27] Marco Kuhrmann, Philipp Diebold, Jürgen Münch, Paolo Tell, Vahid Garousi, Michael Felderer, Kitija Trektere, Fergal McCaffery, Oliver Linssen, Eckhart Hanser, et al. Hybrid software and system development in practice: waterfall, scrum, and beyond. In *Proceedings of the 2017 international conference on software and system process*, pages 30–39, 2017.

[28] RMDH LAVANYA. Green scrum model: Implementation of scrum in green and sustainable software engineering. *International Research Journal of Engineering and Technology*, pages 1583–1587, 2020.

[29] Mou Mahmood, Prangon Chowdhury, Rahbaar Yeassin, Mahmudul Hasan, Tanvir Ahmad, and Nahid-Ur-Rahman Chowdhury. Impacts of digitalization on smart grids, renewable energy, and demand response: An updated review of current applications. *Energy Conversion and Management: X*, page 100790, 2024.

[30] Sara S Mahmoud and Imtiaz Ahmad. A green model for sustainable software engineering. *International Journal of Software Engineering and Its Applications*, 7(4):55–74, 2013.

[31] Keith James Mann, Thomas E Murphy, Nathan Wilson, and Mike West. Magic quadrant for enterprise agile planning tools. *Gartner, Inc*, (G00331978), 2018.

[32] Eric Masanet, Arman Shehabi, Nuoa Lei, Sarah Smith, and Jonathan Koomey. Recalibrating global data center energy-use estimates. *Science*, 367(6481):984–986, 2020.

[33] Shahid Mujtaba, Robert Feldt, and Kai Petersen. Waste and lead time reduction in a software product customization process with value stream maps. In *2010 21st australian software engineering conference*, pages 139–148. IEEE, 2010.

[34] San Murugesan. Harnessing green it: Principles and practices. *IT professional*, 10(1):24–33, 2008.

[35] Stefan Naumann, Markus Dick, Eva Kern, and Timo Johann. The greensoft model: A reference model for green and sustainable software and its engineering. *Sustainable Computing: Informatics and Systems*, 1(4):294–304, 2011.

[36] M Steven Palmquist, Mary Ann Lapham, Suzanne Miller, Timothy Chick, and Ipek Ozkaya. Parallel worlds: Agile and waterfall differences and similarities. *Software Engineering Institute*, 1(1):1–3, 2013.

[37] Kai Petersen and Claes Wohlin. Measuring the flow in lean software development. *Software: Practice and experience*, 41(9):975–996, 2011.

[38] Hezlin Aryani Abd Rahman, Yap Bee Wah, Haibo He, and Awang Bulgiba. Comparisons of adaboost, knn, svm and logistic regression in classification of imbalanced dataset. In *Soft Computing in Data Science: First International Conference, SCDS 2015, Putrajaya, Malaysia, September 2-3, 2015, Proceedings 1*, pages 54–64. Springer, 2015.

[39] Pilar Rodríguez, Mika Mäntylä, Markku Oivo, Lucy Ellen Lwakatare, Pertti Seppänen, and Pasi Kuvaja. Advances in using agile and lean processes for software development. In *Advances in computers*, volume 113, pages 135–224. Elsevier, 2019.

[40] Muhammad Salam and Siffat Ullah Khan. Developing green and sustainable software: Success factors for vendors. In *2016 7th IEEE International Conference on Software Engineering and Service Science (ICSESS)*, pages 1059–1062. IEEE, 2016.

[41] Tridibesh Satpathy et al. A guide to the scrum body of knowledge (sbok™ guide). *Scrumstudy™, a brand of VMEdu, Inc*, page 58, 2016.

[42] Hatef Shamshiri, Ashok Tripathi, Shola Oyedeji, and Jari Porras. Exploring the experiences of experts: Sustainability in agile software development–insights from the finnish software industry. *arXiv preprint arXiv:2407.06978*, 2024.

[43] Shekhar Tankhiwale and Sachin Saraf. Value stream mapping (vsm) led approach for waste and time to market reduction in software product development process. *Telecom Business Review*, 13(1):27, 2020.

[44] M VenkataSwamy, Hrushikesha Mohanty, and Srini Ramaswamy. Vsm: A visual tool for the design and deployment of security requirements. In *2011 Annual IEEE India Conference*, pages 1–6. IEEE, 2011.

[45] Colin C Venters, Rafael Capilla, Stefanie Betz, Birgit Penzenstadler, Tom Crick, Steve Crouch, Elisa Yumi Nakagawa, Christoph Becker, and Carlos Carrillo. Software sustainability: Research and practice from a software architecture viewpoint. *Journal of Systems and Software*, 138:174–188, 2018.

[46] Nina Wolfram, Patricia Lago, and Francesco Osborne. Sustainability in software engineering. *2017 Sustainable Internet and ICT for Sustainability (SustainIT)*, pages 1–7, 2017.

[47] Jamaiah Yahaya, Siti Ibrahim, Komeil Raisian, and Aziz Deraman. Green software process based on sustainability dimensions: The empirical investigation. In *Proceedings of the 1st International Conference on Informatics, Engineering, Science and Technology, INCITEST 2019, 18 July 2019, Bandung, Indonesia*, 2019.