مجلة جامعة صنعاء للعلوم التطبيقية والتكنولوجيا
Sana'a University Journal of Applied Sciences and Technology
https://journals.su.edu.ye/index.php/jast/

Systematic Review of Agile Development Methodologies: Practices, Benefits, Implementation Challenges, and Future Directions

Rania Aleryani[1], Ayman Alsabry[2] * and Emad Alramada[1]

[1]**International University of Technology Twintech, Sana'a, Yemen.,**
[2]**Faculty of Computer Science & Information Technology, International University of Technology Twintech, Sana'a, Yemen.**

*Corresponding author: aymanalsabri@iutt.edu.ye

Abstract

This systematic review synthesizes findings from 47 peer-reviewed studies to comprehensively assess the core practices, benefits, implementation challenges, and future directions of Agile software development methodologies. The review covers a wide range of frameworks, including Scrum, Kanban, Extreme Programming (XP), Lean, Crystal, DSDM, Feature-Driven Development (FDD), Agile Unified Process (AUP), and Behavior-Driven Development (BDD). Using thematic and comparative analysis, the study identifies recurring practices such as sprint planning, continuous integration, pair programming, WIP limits, and test-driven development, which significantly contribute to improved software quality and delivery efficiency. The results indicate that Agile methodologies consistently enhance team collaboration, stakeholder engagement, and adaptability to changing requirements, especially in dynamic environments. Case studies from sectors such as education, healthcare, government, and e-commerce further demonstrate the practical benefits and adaptability of these approaches. However, organizations face several persistent challenges, including cultural resistance, lack of skilled practitioners, stakeholder misalignment, and scalability limitations when applying Agile at scale. The review emphasizes the importance of context-aware methodology selection and customization, and highlights a growing shift towards hybrid models that combine Agile with traditional approaches to better address complex, high-risk, or regulated projects. Finally, the study identifies emerging trends such as Agile–DevOps integration, the application of Agile beyond software development, and the use of technology-driven collaboration and automation tools. These findings provide a practical foundation for practitioners and researchers seeking to optimize Agile adoption and sustain long-term development success.

Article Info

Keywords:
Agile Development, Scrum , Kanban, Extreme Programming (XP), Lean Methodology, Software Engineering, Adaptability, Collaboration, Software Quality, Implementation Challenges

Article History:
Received: 21-June-2025,
Revised: 21-July-2025,
Accepted: 27-July-2025,
Available online: 28 December 2025.

1. INTRODUCTION

Software engineering is defined by IEEE as a systematic, disciplined, and quantifiable approach to software development, operation, and maintenance [1]. It involves management activities, such as planning, coordinating, measuring, monitoring, controlling, and reporting [2, 3]. IEEE, the world's largest technical professional society, plays a crucial role in developing software engineering standards through its Computer Society and Standards Association [4]. Software engineering is inherently co-operative and involves people in development, use, and interaction [5, 6]. This human aspect has become increasingly significant in recent years because of changes in the software domain [7]. The software engineering process encompasses technical, business, and societal contexts, with key activities including requirement gathering, design, construction, testing, and maintenance. These processes are guided by IEEE/EIA 12207.0-1997 standards and the Software Engineering Body of Knowledge (SWEBOK) [8]. Agile software development has revolutionized the manner in which software is planned, built, and delivered. Unlike traditional plan-driven models, Agile emphasizes adaptability, iterative progress, stakeholder collaboration, and continuous feedback, making it a dominant approach in today's fast-paced and uncertain software environments (See Table 1). The growing demand for faster releases, continuous value delivery, and user-centered design has driven organizations across industries to adopt agile practices at scale. Frameworks such as Scrum, Kanban, Extreme Programming (XP), Lean, Crystal, DSDM, Feature-Driven Development (FDD), Agile Unified Process (AUP), and Behavior-Driven Development (BDD) represent diverse approaches within the Agile family. These methodologies offer flexible mechanisms for managing complexity, enhancing team communication, and aligning software development with evolving stakeholder needs (Table 11). Despite their widespread adoption, agile methodologies present several challenges. These include cultural resistance, lack of skilled practitioners, scalability issues, and difficulties aligning practices with organizational goals and regulatory constraints. Moreover, the abundance of frameworks creates ambiguity when selecting the most suitable methodology for specific project contexts (Table 15).

To address these gaps, this systematic review synthesized evidence from 47 peer-reviewed studies to evaluate and compare the practices, advantages, limitations, and implementation challenges of agile methodologies. The study was guided by the following research question:

1) What are the core practices of each Agile methodology?

2) What are the primary benefits of each Agile methodology?

3) What are the key limitations or drawbacks of each Agile methodology?

4) What challenges arise when implementing these methodologies in various contexts?

By answering these questions, this review provides practical, evidence-based insights for researchers, software teams, and decision-makers seeking to adopt or optimize agile development in diverse project environments.

2. BACKGROUND: COMPARISON BETWEEN TRADITIONAL AND AGILE SOFTWARE DEVELOPMENT METHODOLOGIES

Traditional software development methods refer to established practices and frameworks that have been widely used in industry for many years. These methods emphasize a structured approach to software development, often following linear and sequential process [9, 10]. Some of the key traditional methods are as follows:

- The Waterfall model, a traditional software development approach, is characterized by sequential phases, including requirements, design, implementation, verification, and maintenance [11]. While it offers simplicity and clarity for projects with well-defined requirements, its inflexibility to changes has led to the adoption of hybrid approaches combining waterfalls with agile methodologies such as Scrum [12]. The Waterfall method remains prevalent in certain contexts, such as web-based information systems for student admissions, which can improve the efficiency and accuracy [13]. Surprisingly, even in emerging fields such as digital twin modeling, sequential approaches dominate, with hybrid models gaining importance in addressing the diverse requirements of hardware, software, and data management [14]. This trend suggests the continued relevance of waterfall-inspired methodologies, albeit often integrated with more flexible approaches, to meet evolving project needs.

- The V-Model, an extension of the Waterfall model, emphasizes verification and validation at each development stage [11]. This offers a better testing focus and clear relationships between the development and testing phases. However, it remains inflexible and can be costly if changes occur late in the process [11]. Verification and validation (V&V) are critical for ensuring the quality and reliability of software process simulation models [15]. In the medical device industry, an enhanced Agile V-Model has been proposed to address regulatory compliance while maintaining agility [16]. Model-Based Systems Engineering (MBSE) enables the early V&V of system behavior, allowing for analyses prior to implementation. SysML is widely used for system description in MBSE, but V&V solutions tend to be context-specific and face challenges in readiness, handling simplifications, and tool integration [17].

- Incremental development approaches are gaining prominence in software engineering and business model innovations. In software product lines, incremental testing using event-based models can improve fault detection and reduce test generation effort [18]. For business model innovation, an iterative process model with six phases has been proposed to help incumbents adapt to changing market conditions [19]. In global software development, a Software Integration Model (SIM) has been developed to address integration challenges and assess vendor capabilities [20]. For deep learning models, a cost-effective incremental approach (CE-IDM) has been introduced to address capacity scalability, sustainability, and demand challenges in streaming data scenarios. This approach adaptively selects instances for labeling, updates the model structure, and mitigates forgetting, thus outperforming state-of-the-art methods [21].

- The Spiral Model is an important software development methodology that combines iterative development with systematic aspects of the waterfall model [22]. It focuses on risk assessment and management, making it flexible and adaptable to project needs [23]. The model consists of four phases, planning, risk analysis, engineering, and evaluation, which are repeated in spirals [11]. While the Spiral Model offers advantages such as strong risk management and flexibility, it can be complex to manage and requires expertise [24]. This model is particularly useful for large, high-risk projects and has been applied in various contexts, including the implementation of Capability Maturity Model Integration (CMMI) in small organizations [23]. The Spiral Model represents an important evolution in software development methodologies, addressing the limitations of earlier models and emphasizing risk management throughout the development process [22].

- Rapid Application Development (RAD) is an iterative software development approach that emphasizes quick prototyping and user feedback. Recent studies have explored RAD's application of RAD in various contexts, demonstrating its effectiveness in the rapid production of usable systems. RAD has been successfully employed to develop mobile apps for psychiatric patients [25], e-commerce websites [26], and student admission systems [27]. The method typically involves four phases: requirement planning, user design, construction, and cutover [25]. RAD's advantages of RAD include rapid development, enhanced user feedback, and adaptability to urgent needs, such as during the COVID-19 pandemic. However, this may have limitations for large-scale projects. To address potential shortcomings, researchers have proposed integrating RAD with other methodologies, such as Participatory Design, to ensure usability while maintaining rapid development [28]. Traditional software development methodologies are characterized by a structured approach that is well suited for projects with clearly defined requirements and limited anticipated changes. These methodologies share several key characteristics.

- ***Sequential Phases:*** Development progresses through a fixed sequence of stages—typically requirement gathering, design, implementation, testing, and maintenance—where each phase must be completed before the next begins.

- ***Extensive Documentation:*** Comprehensive documentation is produced at each stage to ensure clarity and facilitate maintenance.

- ***Upfront Planning*:** Detailed planning and analysis are conducted at the project's outset to define the scope, timelines, and resources.

- ***Limited Flexibility:*** Once phase is completed, revisiting it is challenging, making these methodologies less adaptable to changing requirements.

- ***Emphasis on Testing Post-Development****:* Testing is typically performed after the implementation phase, which can delay defect identification.

- ***Client Involvement Primarily at Milestones****:* Clients are usually engaged during key milestones, such as initial requirement gathering and final delivery, with limited interaction during intermediate phases.

Agile software development is a lightweight, iterative approach that has gained popularity in modern software engineering [29, 30]. It emphasizes flexibility, frequent reassessment, and adaptation to changing requirements in contrast to traditional plan-based methodologies [29, 31]. Agile methods aim to deliver software features in short iterations, reducing overhead and costs, while accommodating changes at any stage [31]. Popular agile methodologies include Scrum, Extreme Programming, Kanban, and Lean [32]. The agile approach is particularly relevant in cloud computing and big data [33, 34]. However, it has limitations such as restricted attention to structured and architectural design, making it more suitable for small-to-medium design decisions [35]. Selecting the most appropriate agile methodology depends on the specific project requirements, product sensitivity, and organizational structure [36].

As shown in Table 1, Agile and Traditional methodologies differ significantly. Traditional methods use a predictive command-and-control approach, which is ideal for large projects with clear and stable requirements. They emphasize extensive documentation and long-term planning, but lack flexibility, making changes disruptive once projects are underway.

In contrast, agile methodologies adopt an adaptive approach, emphasizing customer collaboration, flexibility, and iterative cycles. For small to medium projects, Agile allows for rapid requirement changes and quick responses to user needs. It fosters collaboration and fast decision-making, and prioritizes the delivery of working software incrementally with minimal documentation.

Larger organizations favor traditional methods for managing extensive, structured projects, while smaller, innovative firms prefer agile for adaptability. The choice depends on the nature, structure, and goalsof the project: agile suits dynamic environments, while traditional excels in stable, large-scale scenarios. Aligning methods with project needs ensures effectiveness.

The choice between agile and traditional approaches depends on multiple factors, including project complexity, regulatory requirements, risk level, and organizational maturity. In many modern contexts, hybrid models that

Table 1. Agile Development vs Traditional Development

Parameter	Traditional Methods	Agile Methods
Ease of Modification	Hard	Easy
Development Approach	Predictive	Adaptive
Development Orientation	Process Oriented	Customer Oriented
Project Size	Large	Small or Medium
Planning Scale	Long Term	Short Term
Management Style	Command and Control	Leadership and Collaboration
Learning	Continuous Learning while Development	Learning is secondary to Development
Documentation	High	Low
Organization Type	High Revenue	Moderate and low Revenue
Organization's Number of Employees	Large	Small
Budget	High	Low
Number of Teams	Multiple	One
Team Size	Medium	Small

integrate both agile and traditional principles have been increasingly adopted to balance flexibility with structure.

3. CASE STUDIES AND REAL-WORLD APPLICATIONS

Several recent studies have focused on the use of agile methodologies in various domains and contexts, highlighting both their benefits and challenges. Hermawan and Indriani (2023) investigated the Agile Kanban method in designing an e-commerce web application for the Friends of Digital Marketing Bandung. This study addressed marketplace limitations, such as branding restrictions and inadequate customer data access, by implementing a tailored solution using PHP and Laravel 9. Kanban's clear task visualization and flexible priority management facilitated improved team collaboration. Despite the time management challenges, usability testing achieved a satisfactory score of 74.66%, demonstrating the effectiveness of the system in reducing marketplace dependency and enhancing operational flexibility [37]. Oliveira et al. (2023) assessed scrum implementation in six federal universities in Brazil. Scrum agility significantly improved communication, collaboration, and productivity while reducing risks and project timelines. However, challenges, such as team turnover and aligning scrum with institutional goals, have been noted. With 96% positive feedback, the study confirmed scrum's potential to optimize IT processes in public institutions, emphasizing the importance of managerial support for successful adoption [38].

Similarly, Fruhling, McDonald, and Dunbar (2008) explored the integration of eXtreme programming (XP) into a U.S. government project for the Strategic Knowledge Integration Web (SKIWeb) system. XP practices, such as on-site collaboration and incremental planning, led to faster delivery and improved satisfaction despite resistance to pair programming and scope creep. The study highlighted the importance of communication and managerial focus when balancing agile and traditional methodologies in government settings [39]. Focusing on lean-agile waste management, Alahyari, Gorschek, and Berntsson Svensson (2019) identified ten categories of waste across 14 Swedish companies, with "task-switching" as the most critical. Using interviews, the study emphasized the need for a holistic, organization-wide approach to waste elimination using tools such as Value Stream Mapping (VSM) and lean principles, addressing fragmented efforts and inconsistent definitions [40].

Firdaus, Ghani, and Yasin (2013) examined Feature Driven Development (FDD) in integrating security requirements within an academic setting. The study found that FDD is adaptable to evolving security needs, such as password encryption, but identified heavy documentation and late-stage changes as challenges. Recommendations include early security integration, iterative refinement, and specialized security roles to better align FDD with secure software practices [41].

In software engineering education, Lin et al. (2013) introduced the Goal-Oriented Agile Unified Process (GOAUP) at Beihang University, China. By aligning hierarchical goals with iterative development, GOAUP demonstrated improved productivity and artifact quality, especially for part-time students with industrial experience. The study underscored the value of goal-net diagrams and adaptive planning in fostering effective software practices in educational contexts [42].

Finally, Scandaroli et al. (2019) explored Behavior-Driven Development (BDD) in two projects by Daitan. BDD improved communication, test coverage, and product alignment in distributed teams despite initial resistance and time demands. Tools such as Cucumber and JBehave bridged gaps between business and technical teams, enhancing software quality, and reducing ambiguity. The study emphasizes that active stakeholder engagement is essential for successful BDD adoption [43].

Collectively, these studies demonstrate the versatility of agile methodologies across domains including e-commerce, education, government systems, and distributed teams. While agile methods such as Kanban, Scrum, XP, FDD, GOAUP, and BDD offer significant benefits, such as improved collaboration, productivity, and software quality, they also reveal challenges, including stakeholder resistance, technical alignment, and managing evolving requirements. These insights underscore the importance of tailoring agile practices to specific project needs, ensuring stakeholder engagement and balancing flexibility with structured processes.

4. METHODOLOGY

This systematic review aims to explore and synthesize studies (S1–S47) that focus on various Agile Development Methodologies and their practices, benefits, and implementation challenges. The study identification and selection process is shown in Figure 1. The methodology consists of the following steps:

1) Study Selection Criteria

The studies included in this review were selected based on the following criteria:

- Relevance: Articles must focus on agile methodologies and their applications in software engineering, including practices, benefits, challenges, and implementation.
- Methodology: Only empirical research, case studies, and theoretical investigations were considered.
- Publication Source: Studies must be published in peer-reviewed journals, academic conferences, or scholarly repositories.
- Publication Date: Only the studies published between 2004 and 2024 were included. Publications outside this date range were excluded from the final analysis to ensure relevance to contemporary agile practices.
- Language: Only studies published in English were considered.

2) Search Strategy

To identify the relevant studies, a structured search was conducted using multiple academic databases. The databases searched were as follows:

· IEEE Xplore
· Scopus
· ScienceDirect
· ACM Digital Library
· SpringerLink
· Google Scholar

The following Boolean search queries were used across all databases:

- "Agile software development" AND ("Scrum" OR "Kanban" OR "Extreme Programming" OR "XP" OR "Lean" OR "Crystal")
- "Agile methodology" AND ("benefits" OR "challenges" OR "implementation")
- "Agile frameworks" AND ("case study" OR "empirical study" OR "software engineering")
- "Hybrid agile" OR "agile vs traditional development"
  The search was limited to studies published in English between 2004 and 2024. Duplicates were removed and only peer-reviewed sources were retained for the final analysis.

3) *Data Extraction and Categorization*

A standardized data extraction form was developed to collect key information from each included study. Two reviewers independently extracted data regarding the following elements:

- Agile methodology discussed (e.g., Scrum, Kanban, XP, Lean, Crystal, etc.)
- Study type (empirical research, case study, or theoretical analysis)
- Sector or domain (e.g., healthcare, education, government, e-commerce)
- Reported practices (e.g., sprint planning, daily stand-ups, pair programming, and test-driven development)
- Stated benefits (e.g., improved delivery speed, team collaboration, software quality)
- Identified challenges (e.g., cultural resistance, scaling difficulties, and lack of skilled practitioners)
- Context or region (geographical setting, organizational environment, or project scale)

Disagreements between the reviewers were resolved through discussion and consensus. The extracted data

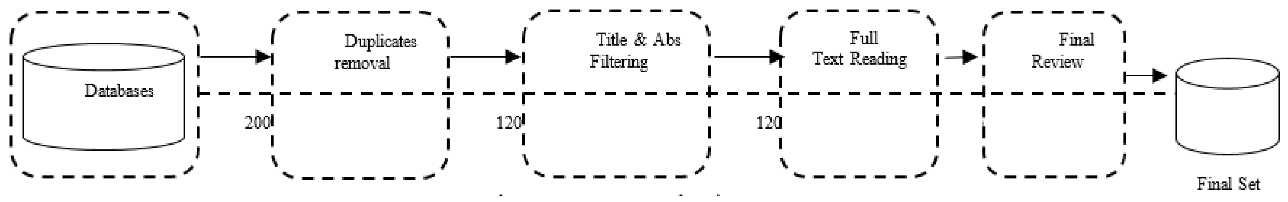

Figure 1. papers selection process

were then categorized thematically using inductive coding to identify recurring patterns and trends across the included studies.

4) Synthesis and Analysis

The analysis employed both qualitative and quantitative methods:

- Thematic Analysis: Extracting themes related to agile practices, team dynamics, and organizational impact.
- Comparative Analysis: Comparing methodologies across studies to identify patterns, commonalities, and unique approaches.

5) Quality Assessment

The quality of the studies was assessed based on methodological rigor, clarity of reporting, and relevance to agile practices. Studies lacking empirical evidence or clear methodologies were excluded to ensure robustness.

6) Reporting Framework

These findings align with the objectives of this review. Each methodology was reviewed individually, summarizing its practices (e.g., sprints and backlog management), reported benefits, and challenges encountered during implementation.

5. AGILE METHODOLOGIES PRACTICES

Agile methodologies have transformed the software development landscape by promoting flexibility, collaboration, and iterative progress. This section explores the diverse practices associated with various agile frameworks, including Scrum, Kanban, Extreme Programming. Each methodology introduces unique principles and techniques that enhance team dynamics, improve project outcomes, and adapt to changing requirements. By examining these practices, we aim to provide a comprehensive understanding of how agile approaches facilitate effective project management and delivery in today's fast-paced development environment.

1) Scrum Methodology

In this section, seven studies (S1-S7) related to scrum method practices are discussed in detail in Table 2. All scrum practices are summarized in Table 11.

Sprints are the foundation of scrum, enabling incremental progress through time-boxed iterations (1–4 weeks). Each sprint begins with Sprint Planning, in which the team sets goals, selects prioritized Product Backlog items, and ensures realistic commitments through effort estimation and capacity planning [S3, S1].

Daily Stand-Ups maintain alignment with short 15-minute meetings focused on progress, plans, and impediments. These meetings foster transparency, problem-solving, and team cohesion and support adaptability, especially in remote settings [S2].

At the sprint's end, the Sprint Review allows the team to present deliverables, gather stakeholder feedback, and refine the Product Backlog to ensure a focus on valuable features [S4]. The Sprint Retrospective reflects on successes and improvements through three key questions: What went well? What didn't? What can improve? This promotes team cohesion and continuous improvement, often by using creative approaches to sustain engagement [S5].

The Product Backlog, maintained by the Product Owner, prioritizes user stories, features, and fixes to align with stakeholder goals. Regular backlog grooming focuses on high-value items, transparency, and efficient resource use [S6]. User Stories, structured as "As a [role], I want [goal] so that [benefit]," clarify the requirements and drive iterative progress. Reviews and retrospectives ensure that user needs remain central, fostering adaptability and continuous improvement [S7].

2) Kanban Methodology

This section discusses studies (S8-S10) related to the Kanban method practices, as outlined in Table 3. A summary of all the Kanban practices is presented in Table 11.

The visual workflow in Kanban enables teams to manage and optimize work processes by visualizing tasks using tools, such as Kanban boards. This promotes transparency, collaboration, and accountability, whereas real-time visibility helps identify and resolve bottlenecks efficiently. A key benefit is limiting Work in Progress (WIP), preventing team overload, and encouraging task completion before new work begins. WIP limits improve workflow transparency, reduce multitasking, and support continuous improvement through retrospectives, fostering a sustainable and efficient pace of work [S8, S9].

Continuous delivery (CD) in Kanban ensures swift and reliable software updates by combining workflow visualization, WIP limits, and automation. By focusing

Table 2. Scrum Method studies

Ref.	Study	Study aim
[44]	S1	The study examines the effectiveness of the Scrum approach in IT project management by identifying benefits and challenges, understanding its adoption in management processes, and evaluating team performance through interviews.
[45]	S2	The study aims to investigate and analyze how Scrum methodologies enhance Agile product development processes, focusing on their implementation practices and benefits.
[46]	S3	The study aims to provide a comprehensive guide for building and implementing the Scrum methodology in organizations
[32]	S4	The study aims to address the research gap concerning the adaptations of Scrum by synthesizing existing research on the modifications proposed to Scrum in various contexts. Specifically, it seeks to answer the research question: "Why and how is Scrum adapted to different contexts?"
[47]	S5	The aim of this study is to investigate how using retrospective games can improve retrospective meetings and potentially lead to positive outcomes for teams.
[48]	S6	The aim of this study is to address the knowledge gap in software-supported Product Backlog (PB) prioritization in Scrum software development projects. The study seeks to develop a conceptual model and a software prototype aligned with the model to enhance PB management by providing methodological support and overcoming communication and collaboration barriers.
[49]	S7	The main aim of this study was to explore and map the existing research on user stories, focusing on research areas, problems addressed, outcomes achieved, research methodologies used, and publication details.

Table 3. Kanban Method studies

Ref.	Study	Study aim
[50]	S8	This study examines how the Kanban model enhances Toyota's organizational agility by identifying key agile characteristics and effective methods for achieving agility. It explores principles like workflow visualization, WIP limits, flow management, and pull systems, offering insights into how Kanban implementation can improve agility.
[51]	S9	This study examines Kanban, Scrum, and Lean methodologies for optimizing production flows, comparing their principles, practices, and strengths. It highlights the importance of selecting the right approach based on production needs, with continuous monitoring and adaptation. The research aims to provide practical insights for improving workflows, productivity, and competitiveness.
[52]	S10	This study implements a Lean Six Sigma-based pull production system in a medical device company to align production with customer demand, reduce waste, and improve supply chain efficiency. Using tools like Value Stream Mapping and Kanban, it aims to cut lead times and inventory by over 60%, mitigating device shortages. The research highlights the benefits of Lean practices in the MedTech industry, addressing gaps in the literature for regulated environments.

on task completion, teams reduce unfinished work and enable frequent and stable deployment. Automation of testing and integration strengthens CD, whereas regular discussions (e.g., stand-ups) align priorities and reinforce agile principles such as adaptability and responsiveness [S9].

The Pull System, central to Kanban, allows work to be "pulled" based on actual demand rather than forecasts. Using visual cues, such as Kanban boards, teams limit WIP, prioritize task completion, and address bottlenecks, enhancing flexibility and efficiency. Rooted in Lean's Kaizen philosophy, the Pull System drives incremental improvements through performance analysis and retrospectives, fostering a culture of responsiveness, collaboration, and sustained value delivery [S10].

3) Extreme Programming (XP)

This section discusses studies (S11-S19) related to the XP method practices, as outlined in Table 4. A summary of all the XP practices is provided in Table 11.

Extreme Programming (XP) is a widely adopted agile methodology that emphasizes flexibility, collaboration, and quality in software development. Several core practices of XP contribute to its success by promoting efficient workflow, enhancing communication, and ensuring the delivery of high-quality software.

The Planning Game in XP fosters collaboration between developers and customers to define project scope and priorities. It begins by gathering and prioritizing user stories based on business value and feasibility, ensuring that critical features are addressed first. Adaptable

Table 4. XP Method studies

Ref.	Study	Study aim
[53]	S11	This study analyzes two agile frameworks, XP and Scrum, exploring their similarities, differences, and complementary attributes. It clarifies their strengths and weaknesses, offering insights for software developers on effectively applying each to meet diverse project needs in modern software development.
[54]	S12	This study explores integrating XP methodologies in global software development (GSD), focusing on overcoming communication challenges in dispersed teams. It analyzes XP's adaptability to GSD, highlighting factors that enhance collaboration and efficiency. The research provides insights for effectively applying XP in global settings to improve software quality and coordination.
[55]	S13	This study examines the role of metaphors in XP to enhance communication and understanding among teams. Using a semiotics-based framework, it identifies methods to create and evaluate system metaphors that clarify architecture and improve collaboration. The research encourages XP teams to adopt metaphors as a key tool for fostering cohesion and efficiency in software development.
[56]	S14	This study explores XP's applications and benefits in software development, highlighting its effectiveness in addressing shifting requirements and enabling rapid, high-quality delivery. Emphasizing customer satisfaction, continuous feedback, and collaboration, it advocates XP's role in improving project management and software quality.
[57]	S15	This study compares Test-Driven Development (TDD) and Behavior-Driven Development (BDD) in enterprise environments, analyzing their impact on delivery speed, software quality, collaboration, and stakeholder satisfaction. Using interviews with developers and managers, it identifies challenges, benefits, and offers insights to help enterprises choose the most suitable approach for their project needs.
[58]	S16	This study explores refactoring practices in microservices and API design, addressing challenges in managing API changes as systems grow complex. Through a survey of architects and developers, it identifies drivers of API changes, quality issues, and proposes a framework for API refactoring to improve service-oriented architecture quality and usability.
[59]	S17	This study examines pair programming (PP) in hybrid work environments, analyzing its dynamics across on-site, remote, and mixed modes. Conducted in a Norwegian fintech company, it offers insights and recommendations for adapting hybrid setups to support effective collaboration and individual preferences.
[60]	S18	This study explores collective ownership in agile IS development teams, examining its role in achieving timely, budget-friendly, and functional system delivery. Using multi-case studies and grounded theory, it analyzes how collective ownership fosters collaboration, shared understanding, and balances individual and team ownership.
[61]	S19	This study systematically evaluates empirical evidence on Continuous Integration (CI) in software development. Through a literature review, it identifies CI's benefits, challenges, and research methodologies, offering insights into its impact on development practices for researchers and practitioners.

to evolving requirements, it integrates user feedback to align it with customer needs. By promoting open communication, the Planning Game builds trust and a shared understanding of project goals [S11].

Small Releases complement the Planning Game by delivering software in incremental functional iterations. This approach provides early access to features, enables continuous feedback, reduces deployment risks, and improves the stability. It enhances user satisfaction by showing tangible progress, fostering continuous improvement, and encouraging innovation without committing to large-scale changes [S11, S12].

In XP, metaphorical practice provides a powerful tool for simplifying complex system concepts. By establishing a shared metaphor, team members—both technical and non-technical—can conceptualize and communicate system architectures in relatable terms. For instance, a system might be likened to a library where data corresponding to books and categories represent shelves. Such metaphors enhance clarity, align development efforts, and encourage creativity in problem solving, fostering collaborative discussions that lead to robust solutions. This practice ensures a shared understanding and bridges communication gaps, contributing to more cohesive and effective teamwork [S11, S13].

The principle of Simplicity in XP underscores the importance of developing straightforward and efficient solutions that meet the current requirements without introducing unnecessary complexity. By focusing on essential features, teams can reduce technical debt and improve code maintainability. Simplified systems are easier to adapt to changes, enabling swift pivots when the requirements evolve. In addition, simplicity enhances team focus, minimizes distractions, and enables steady and efficient progress. This practice cultivates clarity within the codebase, aligns efforts with project goals, and reduces the risk of over-engineered solutions that can hinder long-term maintainability [S11, S14].

Test-Driven Development (TDD) is a cornerstone of XP, ensuring software quality by requiring developers to write tests before implementing the code. The process begins by defining test cases that clarify the expected behavior, followed by writing the minimum code required to pass those tests. This test-first approach helps to identify defects early, fosters continuous validation, and encourages regular code refinement. The TDD also improves maintainability by establishing a comprehensive test suite that safeguards against regressions, enabling confident experimentation and design iterations. This practice promotes reliable software outcomes, while streamlining the development process [S11, S15].

Complementing TDD, Refactoring focuses on enhancing the internal structure of a code without altering its external behavior. As systems evolve, the code can become convoluted, leading to technical debts. Refactoring systematically simplifies code, improves readability, reduces complexity, and ensures alignment with best practice. By fostering clean and maintainable codes, refactoring enhances productivity, reduces long-term risks, and supports continuous improvement. This discipline encourages developers to take ownership of code quality, creating sustainable workflows that accommodate changes without compromising stability [S11, S16].

Pair Programming further reinforces the XP principles by encouraging two developers to collaborate at a single workstation. One developer, the driver, writes the code while the navigator reviews it, offering real-time feedback and strategic insights. This immediate feedback loop reduces errors, enhances code quality, and accelerates problem-solving. Pair Programming also promotes knowledge sharing, reduces knowledge silos, and fosters team cohesion. By working collaboratively, developers improve communication, build trust, and develop a shared understanding of the code base [S11, S17].

The concept of Collective Code Ownership complements Pair Programming by encouraging a shared responsibility for the entire codebase. In this model, any team member can modify any part of the code, eliminate ownership silos, and increase the team flexibility. Collective Code Ownership enhances collaboration, fosters knowledge sharing, and reduces single points of failure, thus ensuring a versatile and adaptable team. This practice supports open communication, accountability, and mutual trust, contributing to higher code quality and team productivity [S11, S18].

Continuous Integration (CI) is another pivotal practice in XP that requires frequent integration and testing of code changes. Developers commit code regularly to a shared repository, where automated builds and tests verify the stability of new contributions. By identifying errors early, the CI reduces integration risks and technical debt while promoting rapid feedback and accountability. This iterative process encourages collaboration as team members remain aware of the system's evolving state, fostering a cohesive and efficient development environment [S11, S19].

The 40-Hour Week principal highlights XP's focus on developer well-being by capping the workweek to avoid burnout. Recognizing that rested developers produce higher-quality work, this practice ensures sustained productivity, clearer problem solving, and improved team morale. By promoting a healthy work-life balance, the 40-Hour Week fosters loyalty, job satisfaction, and innovation, resulting in a sustainable and motivated workforce [S11].

Finally, On-site Customer practice emphasizes close collaboration with a representative customer embedded within the team. This direct engagement ensures the real-time clarification of requirements, immediate feedback, and alignment with user needs. The presence of an on-site customer enhances responsiveness to changing requirements and fosters transparency and trust, creating a partnership dynamic that significantly improves project outcomes [S11].

Code Standards practice ties these principles together by establishing consistent coding guidelines. Shared standards ensure code readability, maintainability, and quality, while reducing ambiguity and fostering team cohesion. By adhering to uniform conventions, developers can efficiently navigate and collaborate on the codebase, streamline workflows, and facilitate effective onboarding for new team members [S11].

4) Lean Software Development

This section discusses studies (S20-S22) related to lean software development practices, as outlined in Table 5. A summary of all the lean software development practices is presented in Table 11.

Lean development focuses on efficiency and customer value by eliminating waste such as unnecessary features, handoffs, incomplete work, and excessive debugging. By addressing inefficiencies across the value stream, teams streamline workflows, reduce costs, and improve responsiveness [S20, S21].

Building quality in the process is essential. Continuous integration (CI) ensures the frequent testing of small changes, enabling early defect detection and reducing later effort. Practices such as test-driven development (TDD) and automated testing further embed quality, leading to reliable products and customer trust [S20, S21, S22].

Rapid delivery through continuous delivery (CD) supports frequent release, allowing real-time feedback and

Table 5. Lean Software Development

Ref.	Study	Study aim
[62]	S20	The study examines lean software development, tracing its roots to Toyota's lean manufacturing. It explores principles like waste elimination, quality focus, and fast delivery, showing their application in software engineering to boost efficiency, collaboration, and customer value. By comparing lean with agile, the study offers insights into the evolution and future of software development.
[63]	S21	The study explores practical Lean implementations in software development, emphasizing goal-oriented, automated measurement to enhance efficiency. It highlights aligning Lean practices with business strategies, addressing team challenges, and institutionalizing experience to reduce reliance on key personnel, optimizing processes for small and medium enterprises.
[64]	S22	The study provides an overview of Agile and Lean methodologies in software development, highlighting their evolution to enhance speed, flexibility, and efficiency. It covers historical trends, theoretical foundations, practical applications, and advancements like continuous delivery and DevOps, offering guidance for effective implementation to improve outcomes.

iterative improvement. Fast delivery cycles align software with evolving market demands, foster continuous learning, and promote adaptability [S20, S21, S22].

Optimizing the whole encourages a system-thinking approach that breaks silos to enhance collaboration and communication. By focusing on the entire value stream, teams can identify bottlenecks, streamline processes, and deliver high-quality outcomes [S20, S21, S22].

Defer Commitment promotes delaying decisions until the "last responsible moment," reducing uncertainty and enabling flexibility. Teams make informed choices that adapt to changing requirements, improving resource allocation and software quality [S20, S21, S22].

Respect People foster a collaborative and empowering culture. Recognizing individual contributions and encouraging shared decision-making boosts team morale, drives innovation, and aligns efforts toward delivering customer value [S20, S21, S22].

Together, these principles create an adaptable value-driven framework that enhances software quality, efficiency, and responsiveness to stakeholder needs.

5) Crystal Methodology

This section discusses studies (S23-S25) related to crystal methodology practices, as outlined in Table 6. A summary of all the crystal practices is presented in Table 11.

Alistair Cockburn's crystal family of agile methodologies provides a flexible, human-centered approach to software development, adapting to team size, project risk, and communication needs.

Crystal methodologies focus on communication, simplicity, and iterative progress to efficiently deliver high-quality software. Crystal Clear, for small teams (2–8), maximizes communication through osmotic interactions, frequent software delivery, and minimal documentation, supported by automated testing [S23]. Crystal Yellow, for medium teams (6–20), emphasizes iterative cycles, face-to-face interactions, user involvement, and simplicity in enhancing productivity and adaptability [S24]. The Crystal Orange for larger teams (10–40) balances agility with structure, promoting frequent delivery, continual feedback, defined roles, and automated testing to ensure scalability and code quality [S25].

Across all Crystal methodologies, Cockburn highlighted adaptability, reflective improvement, and user involvement. For small or large projects, these frameworks prioritize communication, simplicity, and iterative progress to efficiently deliver high-quality software [S23–S25].

6) Dynamic Systems Development Methodology (DSDM)

This section discusses studies (S26-S27) related to DSDM practices, as outlined in Table 7. A summary of all the DSDM practices is provided in Table 11.

The Dynamic Systems Development Method (DSDM) is an agile methodology focused on collaboration, flexibility, and rapid software delivery to meet evolving business needs. Developed in the UK in the 1990s, it ensures high-quality systems within fixed time frames and budgets [S26, S27]. The DSDM emphasizes iterative development, frequent prototyping for user feedback, and active stakeholder involvement. The project lifecycle includes pre-project planning, feasibility analysis, functional design, implementation, and post-project efficiency. Techniques such as timeboxing and prototyping control scopes enable timely adjustments. Although DSDM excels in adaptable, fast-paced projects, it is less suited for rigid, algorithm-heavy, or high-security requirements

Table 6. Crystal Method studies

Ref.	Study	Study aim
[65]	S23	This study on Crystal Clear examined effective practices for small software teams, emphasizing communication, minimal bureaucracy, and rapid quality delivery. It identified key success factors and offered a flexible, adaptable framework for team contexts.
[66]	S24	This study compares agile methodologies—Agile Modeling, XP, Scrum, and Crystal—focusing on their adaptability to dynamic business environments. It analyzes flexibility and responsiveness, offering insights to help developers choose suitable approaches, enhancing decision-making, customer satisfaction, and competitiveness.
[67]	S25	This study analyzes Scrum, Extreme Programming, and Crystal Orange through situated cognition, examining how they support memory, attention, and representation via external artifacts. It highlights their strengths and weaknesses in enhancing cognitive processes, improving teamwork and productivity in dynamic projects.

Table 7. DSDM studies

Ref.	Study	Study aim
[68]	S26	The study explores the Dynamic Systems Development Methodology (DSDM), highlighting its principles of client interaction, iterative prototyping, and flexible requirements to deliver quality software on time and budget. It analyzes DSDM's pros and cons with real-world examples, focusing on its effectiveness in e-commerce and software development challenges.
[69][1]	S27	The study examines the Dynamic Systems Development Method (DSDM) in agile project management, showcasing its blend of traditional and agile practices to improve efficiency and responsiveness. It analyzes DSDM's principles, structure, and techniques, comparing its effectiveness with methodologies like Scrum and Crystal, while highlighting its adaptability to diverse projects.

[S26, S27].

By balancing structured processes with agility, DSDM provides organizations with a robust framework to efficiently develop high-quality software while maintaining alignment with business goals.

7) Feature-Driven Development (FDD)

This section discusses studies (S28-S29) related to FDD practices, as outlined in Table 8. A summary of all the FDD practices is presented in Table 11.
Feature-driven development (FDD) is an agile methodology that focuses on delivering client-valued features through iterative and incremental processes. It includes five steps: overall modeling, feature listing, planning, designing, and building by feature, ensuring quality at each stage. FDD emphasizes team collaboration, clear documentation, and flexibility, making it scalable for medium-to-large projects. Its goal is to deliver high-quality software that meets client expectations and adds value, fostering satisfaction and long-term relationships [S28, S29].

8) Agile Unified Process (AUP)

This section discusses studies (S30-S31) related to AUP practices, as outlined in Table 9. A summary of all the DSDM practices is provided in Table 11.

The Agile Unified Process (AUP) combines Rational Unified Process (RUP) elements with agile principles and offers a flexible framework for projects of varying sizes and complexities. It emphasizes iterative-incremental development in four phases: inception (scope and feasibility), elaboration (refining requirements), construction (coding and testing), and transition (delivery and feedback). Guided by seven workflows, AUP prioritizes collaboration, adaptability, lightweight documentation, and the frequent delivery of working software. Its effectiveness has been demonstrated in real-world applications such as a Service-Oriented Architecture project in a Greek bank, demonstrating its responsiveness and practicality [S30, S31].

9) Behaviour-Driven Development (BDD)

This section discusses studies (S32-S33) related to BDD practices, as outlined in Table 10. A summary of all the BDD practices is presented in Table 11.

BDD is an agile methodology that fosters collaboration between technical and non-technical stakeholders

Table 8. DSDM studies

Ref.	Study	Study aim
[70]	S28	This study explores applying Feature Driven Development (FDD) in developing a Mosque Management Information System. It demonstrates how FDD's iterative, feature-focused approach improves software quality and responsiveness, using a case study to showcase its effectiveness in managing complex, multi-feature projects.
[71]	S29	This study proposes an Enhanced Feature Driven Development (EFDD) model, integrating Behavior Driven Development (BDD) to address FDD's limitations. EFDD improves agility, responsiveness, testing, and continuous delivery, making it ideal for medium and large projects. It enhances communication, documentation, and adaptability in software development.

Table 9. AUP studies

Ref.	Study	Study aim
[72]	S30	The study examines the Agile Unified Process (AUP), a hybrid methodology combining agile practices with the Rational Unified Process (RUP). It highlights AUP's ability to enable incremental releases, enhance collaboration, and adapt to changing requirements while ensuring high-quality software. By detailing its workflows and phases, the study showcases AUP's flexibility and effectiveness across project contexts.
[73]	S31	The study explores project management and methodology development, focusing on innovative technologies and intellectual property. It examines IT integration, neuro-management, and technology transfer strategies to address modern challenges, offering insights to advance academic and practical project management knowledge.

Table 10. BDD studies

Ref.	Study	Study aim
[74]	S32	This study investigates the impact of Behavior Driven Development (BDD) on software processes and quality. It examines BDD's role in improving communication, clarifying requirements, and enhancing collaboration. The research proposes a taxonomy, implementation framework, and future directions to advance BDD practices in software development.
[75]	S33	This study analyzes and compares TDD, BDD, and ATDD, focusing on their objectives, processes, and stakeholder roles. It examines their advantages and disadvantages, highlighting how these methodologies enhance software quality, communication, and alignment with business requirements to improve development practices.

by focusing on software behavior through expected outcomes. Using the "Given-When-Then" format, attention is shifted from implementation to behavior, reducing misunderstandings and improving clarity. The BDD enables automated testing, continuous validation, and living documentation through specifications. While it enhances software quality and stakeholder alignment, challenges include the learning curve, cultural shifts, and tool integration complexity [S32, S33]. Table 11 summarizes all agile development methodologies and their practices, providing brief descriptions.

6. COMPARATIVE ANALYSIS OF AGILE METHODOLOGIES

Comparative analysis of agile methodologies focuses on several key criteria, including focus, core practices, team size, planning and iteration, role of documentation, tools, suitability for project type, strengths, weaknesses, and examples of use cases. This section highlights the differences and similarities among the various agile approaches. A summary of these comparisons is presented in Table 14.

7. BENEFITS OF AGILE DEVELOPMENT METHODOLOGIES

This section discusses studies (S34-S38) on the benefits of agile development methodologies, as outlined in Table 12.

Agile development methodologies enhance software development by prioritizing customer collaboration, improving team dynamics, and enabling continuous improvement through iterative feedback. They support early value delivery with quicker releases, boosting ROI, while their flexibility allows teams to adapt swiftly to changing requirements and market conditions [S34-S38]. The key benefits are summarized in Table 13.

Table 11. Agile Methods Practices

Agile Methods	Practices	Description
Scrum	Sprints	Short, time-boxed periods (usually 2–4 weeks) where a set of tasks is completed.
	Daily Stand-ups	Short, daily meetings where team members discuss progress and obstacles.
	Sprint Planning	A meeting at the beginning of each sprint to plan tasks and set goals.
	Sprint Review	Meeting at the end of each sprint to review completed work and gather feedback.
	Sprint Retrospective	Reflective session after each sprint to discuss improvements.
	Product Backlog	A prioritized list of work, maintained by the Product Owner.
	User Stories	Short, simple descriptions of features from the user's perspective.
Kanban	Visual Workflow	Using boards (physical or digital) to visualize tasks and their progress.
	Limiting Work in Progress (WIP)	Restricting the number of tasks in each stage to improve focus and reduce bottlenecks.
	Continuous Delivery	Continuously delivering small updates or releases instead of time-boxed cycles.
	Pull System	Work is pulled when capacity is available rather than pushed by deadlines.
	Planning Game	A collaborative process where stakeholders—including customers and developers—work together to prioritize user stories and plan upcoming iterations. This practice focuses on aligning development efforts with business goals, estimating effort, and adapting to changing requirements, ensuring that the most valuable features are delivered first.
	Small Releases	Releasing small, incremental updates regularly.
	Metaphor	A shared, simple concept or analogy that describes the system being developed. It helps team members understand the architecture and design by providing a common language and framework. This shared understanding fosters better communication and guides decision-making throughout the development process.

Extreme Programming (XP)	Simplicity (XP)	Focusing on creating only what is necessary for the current iteration.
	Test-Driven Development (TDD)	Writing tests before code to ensure functionality and reduce bugs.
	Refactoring	Continuously improving code without changing its functionality.
	Pair Programming	Two developers work together on the same codebase to improve quality.
	Collective Code Ownership	All developers are responsible for the codebase.
	Continuous Integration	Frequently integrating code changes into a shared repository.
	40-Hour Week	refers to the practice of maintaining a sustainable work schedule, where team members work no more than 40 hours per week. This principle emphasizes the importance of work-life balance, preventing burnout, and ensuring that developers remain productive and engaged over the long term.
	On-site Customer (or Customer Collaboration)	A customer representative is integrated into the team, allowing immediate feedback, clarification, and prioritization. This person helps the team make quick decisions, reducing misunderstandings and rework.
	Code Standards	Adhering to code standards is a fundamental practice in XP and XP2. Standards ensure that code is consistent, readable, and maintainable, making it easier for any team member to work on any part of the code.
Lean Software Development	Eliminate Waste	Reducing unnecessary work, features, or processes.
	Build Quality In	Preventing defects by embedding quality checks throughout the process
	Defer Commitment	Delaying decisions until necessary to reduce rework.
	Deliver Fast	Emphasizing quick iterations and time-to-market.
	Create Knowledge	Emphasize learning, knowledge sharing, and experimentation.
	Optimize the Whole	Focusing on the entire value stream rather than isolated parts
	Respect People	Recognize and leverage the skills, creativity, and expertise of team members.

Crystal Methodologies	Regular Reflective Improvement	Teams regularly assess and improve processes.
	Frequent Delivery	Delivering usable software in regular increments.
	Osmotic Communication	Encouraging informal information flow, especially in collocated teams
	Personal Safety	Creating an environment where team members feel safe sharing ideas
	Focus on Technical Environment	Creating a productive technical and working environment
Dynamic Systems Development Method (DSDM)	MoSCoW Prioritization	Classifying requirements into Must-Have, Should-Have, Could-Have, and Won't-Have
	Timeboxing	Fixing the duration of a phase or task to keep the project on schedule
	Incremental Development	Delivering features in small chunks that can be built upon
	Collaborative Approach	Strong emphasis on team communication and collaboration
	Testing Throughout Lifecycle	Testing is integrated into each phase, not just at the end
Feature-Driven Development (FDD)	Domain Object Modeling	Creating models to represent real-world objects within the project
	Feature Lists	Breaking down projects into small, well-defined features
	Plan by Feature	Prioritizing and scheduling features based on value and feasibility
	Design by Feature	Creating design plans for each feature
	Build by Feature	Developing features individually in small, short iterations
	Regular Builds	Ensuring code is always in a runnable state by compiling often
Agile Unified Process (AUP)	Modeling	Creating visual representations of system requirements and architecture
	Implementation	Building the actual software iteratively
	Testing	Ensuring quality and functionality through consistent testing
	Deployment	Releasing the software in stages
	Configuration Management	Controlling changes to the project artifacts
	Project Management	Using agile management techniques to track and guide the project
	Environment	Providing tools and infrastructure to support development

Behavior-Driven Development (BDD)	Specification by Example	Writing examples of how software should behave based on real user scenarios
	Gherkin Syntax	Defining requirements with Given-When-Then syntax to create standardized acceptance tests
	Automated Testing	Automating tests to ensure scenarios pass and meet the specification
	Collaborative Specification	Teams work together on defining requirements through examples

Table 12. Benefits of Agile Development Methodologies studies

Ref.	Study	Study aim
[76][2]	S34	The study examines the Customer Hierarchy of Needs method in Agile product development, focusing on aligning with customer requirements amid scope creep and changing demands. It details its application at Crown Equipment Corporation, shares lessons learned, and highlights the benefits of prioritizing customer needs for improved satisfaction and outcomes.
[77]	S35	The study evaluates how Agile methodologies enhance team collaboration by improving communication, alignment, and problem-solving, leading to higher productivity and better project outcomes. It analyzes organizational experiences to identify key strategies and challenges in Agile implementation.
[78]	S36	The study explores implementing Continuous Improvement and Flow from Lean principles in Agile projects, providing evidence of their impact on efficiency and real-world benefits. It identifies effective practices and metrics for measuring progress in Agile Continuous Improvement.
[79]	S37	The study explores strategies to enhance early value delivery in agile programs by aligning methodologies with business objectives. Focusing on feedback loops, iterative development, and stakeholder collaboration, it uses case studies to show how organizations can optimize agile practices for timely, meaningful outcomes.
[80]	S38	The study analyzes how agile management principles improve healthcare by enhancing flexibility, responsiveness, and patient outcomes. It highlights benefits like higher patient satisfaction, reduced wait times, and better care quality while addressing adoption challenges.

Table 13. Benefits of Agile Development Methodologies

Benefits of Agile Development Methodologies	Description
Customer-Centric Focus	Prioritizes delivering value to customers by involving them throughout the development process.
Enhanced Team Collaboration	Fosters close cooperation among cross-functional teams, leading to improved communication and shared ownership of the project.
Continuous Improvement	Encourages regular reflection and adaptation, allowing teams to refine processes and enhance product quality over time.
Early Delivery of Value	Facilitates the release of functional product increments early and often, providing immediate value to stakeholders.
Flexibility and Responsiveness	Enables teams to swiftly adapt to changing requirements and market conditions, ensuring the product remains relevant and competitive.

8. IMPLEMENTATION CHALLENGES OF AGILE METHODOLOGIES

Implementing Agile methodologies often encounters several significant challenges that can hinder their effectiveness. These challenges were synthesized into five key categories based on the thematic analysis of the included studies. As shown in Table 15, each challenge was supported by one or more studies (S39–S43) that specifically addressed its characteristics, causes, and potential mitigation strategies. One major issue is cultural resistance to change, where team members and stakeholders may be reluctant to abandon traditional practices in favor of agile principles (S39). Another challenge is the lack of skilled practitioners, as not all teams have access to individuals with adequate training in agile frameworks (S40). The challenge of maintaining quality with speed arises when rapid delivery cycles compromise the software quality (S41). Additionally, managing stakeholder expectations becomes complex in agile environments due to iterative processes and evolving priorities (S42). Finally, scalability issues in large organizations make it difficult to implement agile practices uniformly across diverse teams and departments (S43). Table 15 provides a concise mapping of these challenges to relevant supporting studies.

9. FUTURE DIRECTIONS AND TRENDS IN AGILE DEVELOPMENT

This section shows the studies (S44-S47) related to future directions and trends in agile development methodologies, as outlined in Table 16.

The future of agile development is poised for significant evolution, marked by several emerging trends. Hybrid Agile models are gaining traction as organizations seek to blend traditional and agile methodologies to suit their unique needs and contexts. Moreover, Agile principles are expanding beyond software development, find-

Table 14. Summary of The Comparison of Agile Methodologies

Aspect	Scrum	Kanban	Extreme Programming (XP)	Lean Software Development	Crystal Methodologies	Dynamic Systems Development Method (DSDM)	Feature-Driven Development (FDD)	Agile Unified Process (AUP)	Behavior-Driven Development (BDD)
Focus	Iterative sprints, team collaboration	Continuous flow management	High-quality code and rapid releases	Eliminate waste, optimize processes	Customizable based on team size and project	Business-driven, time and budget focus	Domain modeling and feature delivery	Lightweight adaptation of RUP	Align development with business outcomes
Core Practices	Time-boxed sprints, roles (PO, SM, Team), sprint review	Pull system, WIP limits	Pair programming, TDD, refactoring	Minimize waste, deliver fast	People and interaction focus	MoSCoW prioritization, iterative cycles	Feature list, domain object modeling	Incremental releases, disciplines	Scenarios, given-when-then syntax
Team Size	Small teams (7±2)	Flexible, any size	Small teams, highly collaborative	Any size, focus on efficiency	Small to large, tailored	Medium-sized teams preferred	Small to medium teams	Small to medium teams	Any size, focus on stakeholder collaboration
Planning and Iteration	Sprint planning (2–4 weeks)	Continuous, no iterations	Short iterations (1–2 weeks)	Continuous improvement	Iterative, adaptable	Time-boxed iterations, adjustable scope	Iterative feature modeling	Iterative releases	Iterative feature creation
Role of Documentation	Lightweight, sprint artifacts	Minimal, visual boards	Minimal, code-driven	Minimal, focus on efficiency	As needed, often informal	Just enough for governance	Domain-driven documentation	Essential, but lightweight	Scenarios for clarity
Tools	Scrum board, burndown charts	Kanban board, WIP tracking	Unit tests, CI/CD tools	Value stream maps, flow diagrams	None mandated, team-driven	Project management tools	Domain modeling tools	Simple UML diagrams	Cucumber, SpecFlow
Suitability for Project Type	Complex, evolving requirements	Maintenance and support tasks	Projects with unclear requirements	High variability in workflows	Small to large-scale projects	Business-critical, time-sensitive projects	Complex, domain-specific projects	Medium to large, process-heavy	Projects requiring high collaboration and test-driven clarity
Strengths	Clear roles, structured process	High flexibility, visibility	High-quality code, adaptive	Efficiency, elimination of waste	Customization, team collaboration	Business-focused, structured adaptability	Domain-centric, scalable	Simple, streamlined for RUP users	Clear communication, stakeholder alignment
Weaknesses	Requires discipline, rigidity	Limited guidance, lacks structure	Can be overwhelming for less mature teams	Needs expertise to implement effectively	Overly flexible, less structured	Requires buy-in from business stakeholders	Limited to domain-focused projects	Less guidance than heavier processes	Initial learning curve, tool dependency
Examples of Use Cases	Product development, startups	Operations, DevOps	High-quality software environments	Any process-heavy industries	Varied industries, custom projects	Enterprise-level, time-critical solutions	Domain-driven web applications	Organizations using RUP	High-stake, collaborative software

Table 15. Mapping of Agile Implementation Challenges to Supporting Studies

Ref.	Study	Study aim	Implementation Challenge
[81]	S39	The study examines cultural resistance to change during agile transformations, analyzing how traditional practices and beliefs hinder adoption. It identifies strategies to overcome these barriers and ensure successful agile implementation.	**Cultural Resistance to Change:** Challenges in shifting to an Agile mindset within traditional organizations.
[82]	S40	The study investigates challenges in agile implementation, focusing on the shortage of skilled practitioners. It maps these challenges, examines their characteristics, and compares findings with existing research to suggest ways to enhance practitioner skills and address implementation complexities.	**Lack of Skilled Practitioners:** Issues arising from inadequate training or understanding of Agile principles.
[83]	S41	The study explores how agile practitioners manage quality requirements (QRs) while ensuring rapid delivery. It examines strategies, practices, and challenges, offering insights to improve quality management without compromising speed.	**Maintaining Quality with Speed:** Balancing rapid delivery with high-quality standards.
[84]	S42	The study compares stakeholder expectations—developers and users—regarding user involvement in Agile development. It identifies misalignments that may cause conflicts, aiming to improve communication, user involvement, and project success.	**Managing Stakeholder Expectations:** Challenges in setting realistic expectations in an Agile environment.
[85]	S43	The study examines practices, challenges, and success factors in scaling Agile in large organizations, focusing on key elements for successful implementation and managing multi-team coordination within existing structures.	**Scalability in Large Organizations:** Difficulties in scaling Agile across multiple teams or departments.

ing applications in various sectors such as marketing, education, and project management, thereby broadening its impact. The role of technology is also becoming increasingly prominent, with tools and platforms that enhance collaboration, automation, and real-time feedback driving agile practices. Additionally, the integration of Agile and DevOps fosters a seamless connection between development and operations, promotes continuous delivery, and improves efficiency.

Table 17 provides a summary of these key future directions and trends in agile development, highlighting the dynamic landscape that practitioners will navigate in the coming years [S79-82].

Table 16. Future Directions and Trends in Agile Development

Ref.	Study	Study aim
[86]	S44	The study explores hybrid work in agile software development, examining trends, methodologies, and research gaps. It analyzes how flexible work impacts collaboration, communication, and productivity in software teams.
[87]	S45	The study explores applying Agile methodologies beyond software development, examining their effectiveness in fields like marketing, finance, and project management to enhance flexibility, responsiveness, and value delivery.
[88]	S46	The study explores how technology helps HR foster agile work cultures in the IT sector, promoting innovation and adaptability in fast-changing environments.
[89]	S47	The study explores how integrating Agile and DevOps improves software development by enhancing collaboration, efficiency, and quality.

Table 17. Future Directions and Trends in Agile Development

Future Directions and Trends in Agile Development	Description
Hybrid Agile Models	The emergence of hybrid frameworks combining Agile and traditional practices.
Agile Beyond Software	Expanding Agile principles into non-software industries (e.g., education, government).
Role of Technology	Impact of tools like CI/CD, automation, and digital collaboration tools on Agile practices.
Agile and DevOps Integration	How Agile and DevOps complement each other to accelerate and improve development.

10. CONCLUSION

Agile development methodologies have redefined software engineering by fostering adaptability, collaboration, and efficiency in rapidly changing environments. This review demonstrates that methodologies such as Scrum, Kanban, Extreme Programming (XP), Lean, and Crystal enable teams to deliver high-quality software while responding effectively to evolving requirements and stakeholder expectations. However, the successful implementation of agile practices requires addressing inherent challenges, including resistance to change, inadequate stakeholder alignment, and difficulties in scaling complex or large projects. Organizations must focus on fostering a culture of continuous improvement, integrating agile practices with strategic objectives, and tailoring methodologies to project-specific needs.

Future research should explore hybrid models that combine the strengths of traditional and agile frameworks to handle high-risk large-scale projects with greater precision. Additionally, the growing influence of technologies, such as artificial intelligence and DevOps, in agile contexts presents an opportunity to enhance automation, decision-making, and process efficiency. By embracing a strategic, context-aware approach, organizations can unlock the full potential of agile methodologies, ensuring that their software development processes remain innovative, sustainable, and aligned with dynamic business goals.

REFERENCES

[1] E. Bodden et al., “Ernst denert software engineering award 2022,” in *Ernst Denert Award for Software Engineering 2022: Practice Meets Foundations*, Cham: Springer Nature Switzerland, 2024, pp. 1–8.

[2] H. Edison, X. Wang, and K. Conboy, “Comparing methods for large-scale agile software development: A systematic literature review,” *IEEE Trans. on Softw. Eng.*, vol. 48, no. 8, pp. 2709–2731, 2021.

[3] Y. Shastri, R. Hoda, and R. Amor, “The role of the project manager in agile software development projects,” *J. Syst. Softw.*, vol. 173, p. 110 871, 2021.

[4] R. F. Schmidt, *Software engineering: architecture-driven software development*. Newnes, 2013.

[5] I. Omoronyia, J. Ferguson, M. Roper, and M. Wood, “A review of awareness in distributed collaborative software engineering,” *Software: Pract. Exp.*, vol. 40, no. 12, pp. 1107–1133, 2010.

[6] D. Tang et al., “Collaborative agents for software engineering,” *arXiv preprint arXiv:2402.02172*, 2024.

[7] A. Seffah, J. Vanderdonckt, and M. C. Desmarais, *Human-centered software engineering: Software engineering models, patterns and architectures for HCI*. Springer Science & Business Media, 2009.

[8] R. H. Thayer and M. J. Christensen, *Software Engineering, The Development Process-Volume 1*. Wiley-IEEE Computer Society Press, 2005.

[9] N. Alkıs-Bayhan, E. Özmen, and E. Karaman, “Comparing software development life cycle models with multi-criteria decision making approach,” in *2024 10th International Conference on Control, Decision and Information Technologies (CoDIT)*, IEEE, 2024, pp. 993–998.

[10] D. B. Aniley, E. A. Jalew, and G. A. Agegnehu, “Selection of software development life cycle models using machine learning approach,” *Int. J. Comput. Appl.*, vol. 975, p. 8887,

[11] A. Alazzawi and B. Rahmatullah, “A comprehensive review of software development life cycle methodologies: Pros, cons, and future directions,” *Iraqi J. For Comput. Sci. Math.*, vol. 4, no. 4, pp. 173–190, 2023.

[12] N. Yahya and S. S. Maidin, “Hybrid agile development phases: The practice in software projects as performed by software engineering team,” *Indonesian J. Electr. Eng. Comput. Sci.*, vol. 29, no. 3, pp. 1738–1749, 2023.

[13] G. K. Daha, N. Faizah, and W. Nurcahyo, “Rancangan sistem informasi penerimaan siswa baru smp swasta generasi kedde wali berbasis web dengan metode waterfall,” *Comput. J.*, vol. 1, no. 1, pp. 15–22, 2023.

[14] L. Sommer, “Digital twin modeling: A comparison of current approaches,” *Open Res. Eur.*, vol. 4, no. 56, p. 56, 2024.

[15] Y. Li, H. Zhang, B. Liu, L. Dong, H. Gong, and G. Rong, “Verification and validation of software process simulation models: A systematic mapping study,” *J. Software: Evol. Process.*, e2612, 2024.

[16] A. A. Khan, M. U. Akram, W. H. Butt, and M. Sirshar, “An enhanced agile v-model: Conformance to regulatory bodies and experiences from model’s adoption to medical device development,” *Heliyon*, vol. 10, no. 6, 2024.

[17] J. Cederbladh, A. Cicchetti, and J. Suryadevara, “Early validation and verification of system behaviour in model-based systems engineering: A systematic literature review,” *ACM Trans. on Softw. Eng. Methodol.*, vol. 33, no. 3, pp. 1–67, 2024.

[18] M. Beyazit, T. Tuglular, and D. Ö. Kaya, “Incremental testing in software product lines—an event based approach,” *IEEE Access*, vol. 11, pp. 2384–2395, 2023.

[19] N. Bachmann and H. Jodlbauer, “Iterative business model innovation: A conceptual process model and tools for incumbents,” *J. Bus. Res.*, vol. 168, p. 114 177, 2023.

[20] M. Ilyas, S. U. Khan, H. U. Khan, and N. Rashid, “Software integration model: An assessment tool for global software development vendors,” *J. Software: Evol. Process.*, vol. 36, no. 4, e2540, 2024.

[21] Y. Yang, D.-W. Zhou, D.-C. Zhan, H. Xiong, Y. Jiang, and J. Yang, “Cost-effective incremental deep model: Matching model capacity with the least sampling,” *IEEE Trans. on Knowl. Data Eng.*, vol. 35, no. 4, pp. 3575–3588, 2021.

[22] D. A. Grier, “The outward spiral,” *Computer*, vol. 53, no. 4, pp. 73–76, 2020.

[23] M. R. Ayyagari and I. Atoum, “Cmmi-dev implementation simplified,” *Int. J. Adv. Comput. Sci. Appl.*, vol. 10, no. 4, 2019.

[24] A. Singh and P. J. Kaur, “Analysis of software development life cycle models,” in *Proceeding of the Second International Conference on Microelectronics, Computing & Communication Systems (MCCS 2017)*, Springer, 2019, pp. 689–699.

[25] A. A. Shaker et al., “Facilitating in-house mobile app development within psychiatric outpatient services for patients diagnosed with borderline personality disorder: Rapid application development approach,” *JMIR Hum. Factors*, vol. 10, e46928, 2023.

[26] R. D. Atmaja, N. Faizah, and M. A. Kambry, “Aplikasi e–commerce toko sinar bella dengan metode rapid application development (rad) menggunakan framework codeigniter 4,” *Des. J.*, vol. 1, no. 1, pp. 26–37, 2023.

[27] M. Y. B. Poso, N. Faizah, and P. K. Karo, “Aplikasi sistem penerimaan siswa baru smk taruna bakti cikarang selatan berbasis web dengan metode rapid application development (rad),” *Des. J.*, vol. 1, no. 1, pp. 72–78, 2023.

[28] Y.-A. Daraghmi and E.-Y. Daraghmi, “Rapd: Rapid and participatory application development of usable systems during covid19 crisis,” *IEEE Access*, vol. 10, pp. 93 601–93 614, 2022.

[29] G. Waja, J. Shah, and P. Nanavati, “Agile software development,” *Int. J. Eng. Appl. Sci. Technol.*, vol. 5, no. 12, pp. 73–78, 2021.

[30] A.-M. Gheorghe, I. D. Gheorghe, and I. L. Iatan, “Agile software development,” *Informatica Econ.*, vol. 24, no. 2, 2020.

[31] S. Al-Saqqa, S. Sawalha, and H. AbdelNabi, “Agile software development: Methodologies and trends,” *Int. J. Interact. Mob. Technol.*, vol. 14, no. 11, 2020.

[32] E. C. Daraojimba, C. N. Nwasike, A. O. Adegbite, C. A. Ezeigweneme, and J. O. Gidiagba, “Comprehensive review of agile methodologies in project management,” *Comput. Sci. & IT Res. J.*, vol. 5, no. 1, pp. 190–218, 2024.

[33] B. D. Simpson, E. Johnson, G. S. Adeleke, C. P. Amajuoyi, and O. B. Seyi-Lande, “Leveraging big data for agile transformation in technology firms: Implementation and best practices,” *Eng. Sci. & Technol. J.*, vol. 5, no. 6, pp. 1952–1968, 2024.

[34] F. E. Aouni, K. Moumane, A. Idri, M. Najib, and S. U. Jan, “A systematic literature review on agile, cloud, and devops integration: Challenges, benefits,” *Inf. Softw. Technol.*, p. 107 569, 2024.

[35] M. Nawaz, T. Nazir, S. Islam, M. Masood, A. Mehmood, and S. Kanwal, “Agile software development techniques: A survey,” *Proc. Pak. Acad. Sci. A. Phys. Comput. Sci.*, vol. 58, no. 1, pp. 17–33, 2021.

[36] R. D. Estrada-Esponda, M. López-Benítez, G. Matturro, and J. C. Osorio-Gómez, “Selection of software agile practices using analytic hierarchy process,” *Heliyon*, vol. 10, no. 1, 2024.

[37] I. Hermawan and I. Indriani, “Design get up application e-commerce based web using the agile kanban method (case study at bandung digital marketing friends),” *APPLIED SCIENCE AND TECHNOLOGY RESEARCH JOURNAL*, vol. 2, no. 2, pp. 81–91, 2023.

[38] E. R. de Oliveira, P. C. C. Ribeiro, M. P. Méxas, and S. B. de Oliveira, “Scrum method assessment in federal universities in brazil: Multiple case studies,” *Braz. J. Oper. & Prod. Manag.*, vol. 20, no. 1, pp. 1496–1496, 2023.

[39] A. Fruhling, P. McDonald, and C. Dunbar, “A case study: Introducing extreme programming in a us government system development project,” in *Proceedings of the 41st Annual Hawaii International Conference on System Sciences (HICSS 2008)*, IEEE, 2008, pp. 464–464.

[40] H. Alahyari, T. Gorschek, and R. B. Svensson, “An exploratory study of waste in software development organizations using agile or lean approaches: A multiple case study at 14 organizations,” *Inf. Softw. Technol.*, vol. 105, pp. 78–94, 2019.

[41] A. Firdaus, I. Ghani, and N. I. M. Yasin, “Developing secure websites using feature driven development (fdd): A case study,” *J. Clean Energy Technol.*, vol. 1, no. 4, pp. 322–326, 2013.

[42] J. Lin, C. Miao, Z. Shen, and W. Sun, “Goal oriented agile unified process (goaup): An educational case study,” in *2013 International Conference on Software Engineering and Computer Science*, Atlantis Press, 2013, pp. 36–44.

[43] A. Scandaroli, R. Leite, A. H. Kiosia, and S. A. Coelho, "Behavior-driven development as an approach to improve software quality and communication across remote business stakeholders, developers and qa: Two case studies," in *2019 ACM/IEEE 14th International Conference on Global Software Engineering (ICGSE)*, IEEE, 2019, pp. 105–110.

[44] V. Nuti, "Analysis of the effectiveness of the scrum approach in the management of an it project," Ph.D. dissertation, Politecnico di Torino, 2023.

[45] C. C. Ekechi, C. D. Okeke, and H. E. Adama, "Enhancing agile product development with scrum methodologies: A detailed exploration of implementation practices and benefits," *Eng. Sci. & Technol. J.*, vol. 5, no. 5, pp. 1542–1570, 2024.

[46] O. Zhmai and K. Badera, "Stages of building and implementing the scrum methodology," *Èkon. ta suspilstvo*, no. 42, 2022.

[47] A. Przybyłek, M. Albecka, O. Springer, and W. Kowalski, "Game-based sprint retrospectives: Multiple action research," *Empir. Softw. Eng.*, vol. 27, no. 1, p. 1, 2022.

[48] K. Model and G. Herzwurm, "Software-supported product backlog prioritization in scrum software development projects," in *ICSOB Companion*, 2022.

[49] A. R. Amna and G. Poels, "Systematic literature mapping of user story research," *IEEE Access*, vol. 10, pp. 51 723–51 746, 2022.

[50] I. Hachemi and M. Bakhouche, "The impact of the kanban model on toyota's agility," *Dirassat J. Econ. Issue*, vol. 15, no. 2, pp. 233–248, 2024.

[51] S. Sotnik, M. Omarov, A. Frolov, and B. A. A. Al-Badani, "Optimization of work: In-depth look at kanban, scrum and lean," *J. Nat. Sci. Technol.*, vol. 3, no. 1, pp. 290–301, 2024.

[52] D. O'Donoghue, O. McDermott, A. Trubetskaya, A. Rosa, M. Kharub, and K. Cormican, "Implementing an order pull system into a medical device company using define for lean six sigma methodology," in *European Lean Educator Conference*, Springer, 2023, pp. 306–326.

[53] A. Akhtar, B. Bakhtawar, and S. Akhtar, "Extreme programming vs scrum: A comparison of agile models," *Int. J. Technol. Innov. Manag. (IJTIM)*, vol. 2, no. 2, pp. 80–96, 2022.

[54] T. R. Ojha and P. Chaudhary, "Enabling extreme programming (xp) in global software development (gsd) practice," *J. Adv. Softw. Eng. Test.*, vol. 5, no. 3, 2022.

[55] C. V. P. Krishna, *Design and metaphor evaluation for extreme software development ch v phani krishna, nethravathi ps 2, k bhargavi*.

[56] A. Shrivastava, I. Jaggi, N. Katoch, D. Gupta, and S. Gupta, "A systematic review on extreme programming," in *Journal of Physics: Conference Series*, vol. 1969, IOP Publishing, 2021, p. 012 046.

[57] J. Cui, *A comparative study on the impact of test-driven development (tdd) and behavior-driven development (bdd) on enterprise software delivery effectiveness*, arXiv preprint arXiv:2411.04141, 2024.

[58] M. Stocker and O. Zimmermann, "From code refactoring to api refactoring: Agile service design and evolution," in *Service-Oriented Computing: 15th Symposium and Summer School, SummerSOC 2021, Virtual Event, September 13–17, 2021, Proceedings 15*, Springer, 2021, pp. 174–193.

[59] A. Tkalich, N. B. Moe, N. H. Andersen, V. Stray, and A. M. Barbala, "Pair programming practiced in hybrid work," in *2023 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, IEEE, 2023, pp. 1–7.

[60] S. Nazir, S. E. Collignon, and N. C. Surendra, "Understanding collective ownership in agile development: Turbo charging the process," *Inf. & Manag.*, vol. 61, no. 6, p. 104 004, 2024.

[61] E. Soares, G. Sizilio, J. Santos, D. A. D. Costa, and U. Kulesza, "The effects of continuous integration on software development: A systematic literature review," *Empir. Softw. Eng.*, vol. 27, no. 3, p. 78, 2022.

[62] M. Poppendieck and M. A. Cusumano, "Lean software development: A tutorial," *IEEE Softw.*, vol. 29, no. 5, pp. 26–32, 2012.

[63] A. Janes and G. Succi, *Lean software development in action*. Springer, 2014.

[64] P. Rodríguez, M. Mäntylä, M. Oivo, L. E. Lwakatare, P. Seppänen, and P. Kuvaja, "Advances in using agile and lean processes for software development," in *Advances in Computers*, vol. 113, Elsevier, 2019, pp. 135–224.

[65] A. P. Becker and A. Cockburn, *Crystal Clear: A Human-Powered Methodology for Small Teams*. Pearson Education, 2004.

[66] R. Kumar, P. Maheshwary, and T. Malche, "Inside agile family software development methodologies," *Int. J. Comput. Sci. Eng.*, vol. 7, no. 6, pp. 650–660, 2019.

[67] N. K. Do, *Situated cognition and agile software development: A comparison of three methods*, Unpublished or unspecified source, 2010.

[68] D. D. Teixeira, F. J. A. Pires, J. P. G. de Sousa, and T. A. G. P. S. Pinto, *Dsdm–dynamic systems development methodology*, Faculdade de Engenharia da Universidade do Porto. Available online, URL: http://paginas.fe.up.pt (access details incomplete), 2005.

[69] B. J. Voigt, M. Glinz, and D.-I. C. Seybold, *Dynamic system development method*, Report or internal publication, 2004.

[70] S. R. Riady, K. Sofi, J. Shadiq, and R. W. Arifin, "Selection of feature driven development (fdd) model in agile method for developing information system of mosque management," *J. Comput. Networks, Archit. High Perform. Comput.*, vol. 4, no. 2, pp. 127–136, 2022.

[71] Z. Nawaz, "Proposal of enhanced fdd process model," *Int. J. Educ. Manag. Eng. (IJEME)*, vol. 11, no. 4, pp. 43–50, 2021.

[72] C. Edeki, "Agile unified process," *Int. J. Comput. Sci.*, vol. 1, no. 3, pp. 13–17, 2013.

[73] V. Cherepanova, O. Podrez, and P. Pererva, *Management of international projects using the agile method*, Conference/compilation volume; page 25; text in Ukrainian, 2023.

[74] M. S. Farooq, U. Omer, A. Ramzan, M. A. Rasheed, and Z. Atal, "Behavior driven development: A systematic literature review," *IEEE Access*, 2023.

[75] M. M. Moe, "Comparative study of test-driven development tdd, behavior-driven development bdd and acceptance test–driven development atdd," *Int. J. Trend Sci. Res. Dev.*, vol. 3, no. 4, pp. 231–234, 2019.

[76] P. Dysert and S. Prabhala, "Customer hierarchy of needs: Customer centric approach to agile product development," in *International Conference on Human-Computer Interaction*, Springer, 2022, pp. 3–11.

[77] M. Celestin, S. Sujatha, A. D. Kumar, and M. Vasuki, "The rise of agile methodologies in managing complex business projects: Enhancing efficiency, collaboration, and adaptability," *Indo Am. J. Multidiscip. Res. Rev.*, vol. 8, no. 2, pp. 69–77, 2024.

[78] B. Swaminathan and K. Jain, "Implementing the lean concepts of continuous improvement and flow on an agile software development project: An industrial case study," in *2012 Agile India*, IEEE, 2012, pp. 10–19.

[79] K. Lakshminarasimham, “Enhancing value delivery in agile programs: Strategies and best practices,” *Int. J. Creat. Res. In Comput. Technol. Des.*, vol. 6, no. 6, pp. 1–27, 2024.

[80] H. H. Saleh, Z. A. Abbas, N. Latif, and Z. T. Khalil, “Agile management in healthcare improving patient outcomes through flexibility and responsiveness,” *J. Ecohumanism*, vol. 3, no. 5, pp. 633–649, 2024.

[81] F. Reginaldo and G. Santos, “Challenges in agile transformation journey: A qualitative study,” in *Proceedings of the XXXIV Brazilian Symposium on Software Engineering*, 2020, pp. 11–20.

[82] P. Gregory, L. Barroca, H. Sharp, A. Deshpande, and K. Taylor, “The challenges that challenge: Engaging with agile practitioners’ concerns,” *Inf. Softw. Technol.*, vol. 77, pp. 92–104, 2016.

[83] P. Karhapää et al., “Strategies to manage quality requirements in agile software development: A multiple case study,” *Empir. Softw. Eng.*, vol. 26, no. 2, p. 28, 2021.

[84] J. Buchan, M. Bano, D. Zowghi, S. MacDonell, and A. Shinde, “Alignment of stakeholder expectations about user involvement in agile software development,” in *Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering*, 2017, pp. 334–343.

[85] M. Kalenda, P. Hyna, and B. Rossi, “Scaling agile in large organizations: Practices, challenges, and success factors,” *J. Software: Evol. Process.*, vol. 30, no. 10, e1954, 2018.

[86] D. Khanna, E. L. Christensen, S. Gosu, X. Wang, and M. Paasivaara, “Hybrid work meets agile software development: A systematic mapping study,” in *Proceedings of the 2024 IEEE/ACM 17th International Conference on Cooperative and Human Aspects of Software Engineering*, 2024, pp. 57–67.

[87] G. S. Matharu, A. Mishra, H. Singh, and P. Upadhyay, “Empirical study of agile software development methodologies: A comparative analysis,” *ACM SIGSOFT Softw. Eng. Notes*, vol. 40, no. 1, pp. 1–6, 2015.

[88] F. A. Ajayi and C. A. Udeh, “Agile work cultures in it: A conceptual analysis of hr’s role in fostering innovation supply chain,” *Int. J. Manag. & Entrepreneurship Res.*, vol. 6, no. 4, pp. 1138–1156, 2024.

[89] N. G. Cholli, *Integrating agile and devops: A comprehensive review of devops-enabled agile in modern software development*, Unpublished or unspecified source, 2024.