RESEARCH ARTICLE

Comparing Agile Transitions: A Study of XP, Scrum, and Hybrid Frameworks

Samia Akhtar[1,*] **and Shabib Aftab**[1]

[1] Department of Computer Science, Virtual University of Pakistan, Lahore 54000, Pakistan

Abstract

Agile has become a cornerstone of modern software development. Among its many frameworks, Extreme Programming (XP) and Scrum are the most widely recognized. XP emphasizes technical practices and engineering discipline while Scrum provides structured roles and iterative planning. Over time, many organizations have also adopted hybrid models that combine the strengths of both. Despite their popularity, teams often face challenges when deciding which approach to adopt. The choice between XP, Scrum or a hybrid is not always straightforward as each carries different strengths, limitations and suitability for specific contexts. This paper addresses this issue by presenting a comparative analysis of XP, Scrum and their hybrids. First, we revisit their phases and key practices along with their strengths, weaknesses and application. Then a detailed comparison is presented between XP, Scrum and their Hybrids. Building on this analysis, we have proposed a structured decision framework. This framework provides a clear criteria and step-by-step guidance to help practitioners select the most suitable approach for their projects. The framework is supported by subsections that explain evaluation criteria and conceptual use cases. In addition, published case studies are discussed to validate the framework and show how XP, Scrum, and hybrid methods are applied in practice. The paper also outlines future directions for agile practices including the role of AI, scaling strategies and distributed collaboration. Through this work, the paper offers both critical insights and practical tools for researchers and practitioners. It highlights not only how XP and Scrum compare but also how hybrid approaches can improve agile adoption in today's dynamic development landscape.

Keywords: extreme programming, scrum, agile methodologies, comparative analysis, hybrid models.

1 Introduction

Software development has always required structured methods to manage complexity, reduce risk and deliver quality results [1]. Early approaches such as the Waterfall model provided clear phases, detailed documentation and predictable planning. These qualities gave managers control but they also created problems when customer needs or market conditions changed. Long cycles made it difficult to adapt when projects often ended up delayed, over budget, or misaligned with user expectations [2]. To

Submitted: 06 September 2025
Accepted: 11 December 2025
Published: 08 February 2026

Vol. 2, **No.** 1, 2026.
10.62762/JSE.2025.428569

***Corresponding author:**
Samia Akhtar
samiaakhtar9898@gmail.com

Citation
Akhtar, S., & Aftab, S. (2026). Comparing Agile Transitions: A Study of XP, Scrum, and Hybrid Frameworks. *ICCK Journal of Software Engineering*, 2(1), 30–51.

© 2026 by the Authors. Published by Institute of Central Computation and Knowledge. This is an open access article under the CC BY license (https://creativecommons.org/licenses/by/4.0/).

address these weaknesses, iterative and incremental methods appeared offering feedback at shorter intervals and allowing course corrections during development. This evolution eventually led to agile, which became a defining philosophy in modern software engineering [3]. Formally introduced in 2001 through the Agile Manifesto, agile shifted the focus from rigid planning to flexibility, collaboration and customer value. It emphasized small frequent deliveries and continuous feedback loops [4]. Over the years, agile expanded beyond software to industries such as healthcare, finance, education and government. Its ability to reduce risk, improve collaboration and align teams with user needs fueled its adoption. At the same time, challenges emerged [5]. Agile requires cultural change and high collaboration, which are not always easy to achieve. Regulated industries demand more documentation than agile naturally provides [6]. Large and distributed teams struggle to maintain consistent practices. Today, agile is further shaped by artificial intelligence, DevOps pipelines, and remote-first environments. These forces have redefined how agile methods are applied and tested their ability to remain adaptable and effective.

Within agile, Extreme Programming (XP) and Scrum have become two of the most recognized and debated frameworks. Both are grounded in agile principles but differ greatly in their emphasis. XP focuses on technical rigor and engineering discipline using practices such as test-driven development, pair programming, continuous integration and frequent releases. Scrum, in contrast defines structured roles, artifacts, and ceremonies to manage work through iterative sprints [7]. This difference has created uncertainty for practitioners and organizations. Some find XP effective for highly technical and fast-changing projects. Others prefer Scrum's structure for managing larger teams and organizational processes. Many adopt hybrids that combine the strengths of both. However, most existing studies either focus on one framework or compare them from an earlier era without considering modern realities such as AI support, DevOps automation, and distributed collaboration [8]. This gap limits guidance for today's practitioners. In this paper, we critically re-examine XP and Scrum. We compare their principles, practices, and applications. We highlight strengths and weaknesses, explore hybrid approaches, and propose a decision framework to guide adoption. By doing so, this paper offers updated insights and practical recommendations for applying agile frameworks in today's evolving software development landscape.

1.1 Scope and Objectives

While Agile as a whole encompasses a variety of frameworks, this paper narrows its focus to XP, Scrum and their hybrids because of their strong influence on modern software development practices. The study positions itself not only as a comparison of established approaches but also as a guide for navigating the growing complexity of agile adoption in modern contexts. By including conceptual insights and practical perspectives, the work bridges the gap between theory and application ensuring its relevance to both researchers and practitioners. The objective of the study is to:

- Revisit and analyze the origins and phases of XP and Scrum to provide both historical grounding and conceptual clarity.
- Analyze each framework's philosophy, practices, advantages, limitations and modern relevance.
- Explore hybrid models of XP and Scrum and identify their common practices. Highlight the benefits and challenges of adopting hybrid approaches.
- Provide a structured comparative analysis of XP, Scrum, and their hybrid models.
- Propose a decision framework that offers criteria and step-by-step guidance for selecting or tailoring XP, Scrum or hybrid approaches.
- Support the framework with conceptual examples and real-world case studies to ensure its practical relevance.
- Outline future directions for agile methodologies emphasizing scalability, intelligent automation and collaboration in global settings.

1.2 Paper Organization

The rest of this paper is divided into eight sections. Section 2 explains the context of agile in the modern era. It describes the origins of agile, its evolution over time, and the role of artificial intelligence in shaping agile practices. Section 3 focuses on Extreme Programming (XP). It discusses its phases, core philosophy, key practices, applications, and relevance in modern projects. Section 4 presents Scrum. It explains the framework, values, practices, applications, and its continued importance. Section 5

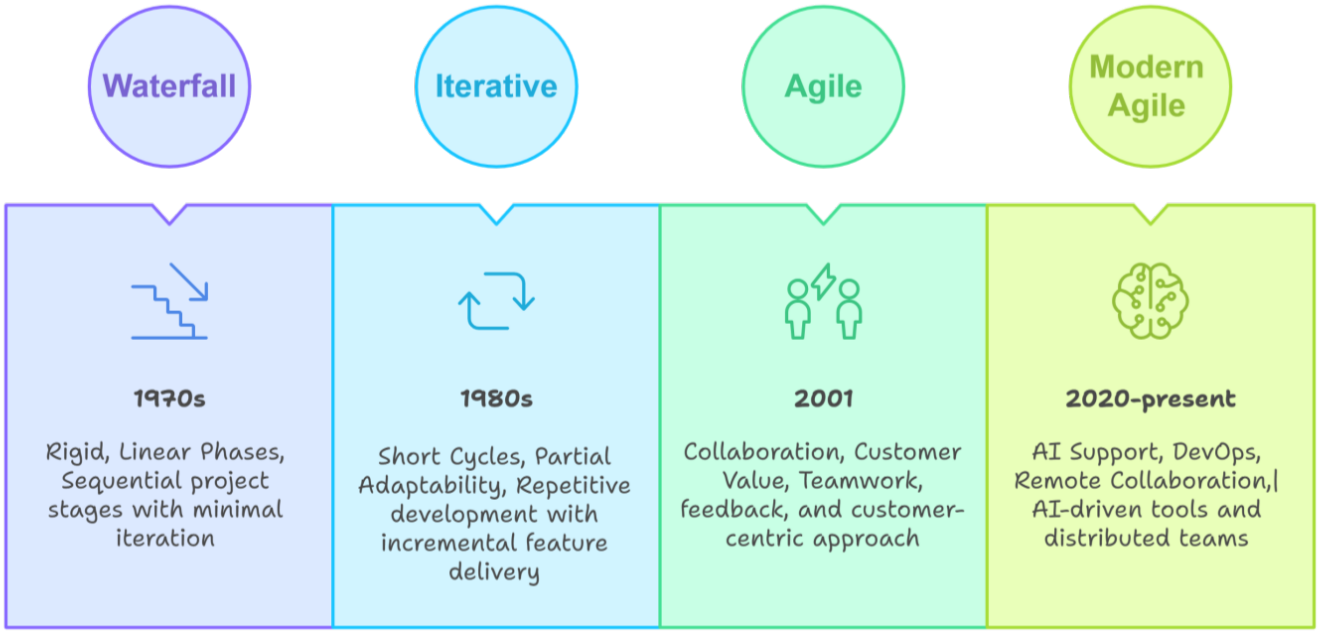


Figure 1. Evolution of software development methodologies from Waterfall to Modern Agile.

talks about Hybrid of XP and Scrum, their common practices, benefits and limitations. Section 6 provides a comparative analysis of XP, Scrum and hybrid models. It highlights their similarities, differences, and integration with modern tools and methods. Section 7 introduces a decision framework. It guides practitioners in selecting or adapting XP, Scrum, or hybrid approaches and includes examples from case studies. Section 8 explores future directions for agile practice. It emphasizes the influence of AI, DevOps, distributed teamwork, and scaling strategies. Section 9 concludes the paper by summarizing the findings. It also highlights the contributions of the study and provides practical guidance for both practitioners and researchers.

2 Context of Agile in the Modern Era

Agile has moved far beyond its origins as a lightweight alternative to traditional software development [9]. Over the past two decades, it has matured into a dominant philosophy shaping not only how software is built but also how organizations manage change, collaboration and delivery [10]. At the same time, new technologies and shifting work cultures have continuously reshaped how Agile is practiced [11]. To understand the position of Extreme Programming (XP) and Scrum within this evolving landscape, it is essential to examine how Agile itself has transformed the industries it now serves and the challenges it faces in contemporary practice.

2.1 The Origin of Agile Methodologies

The origins of agile methodologies can be traced back to the challenges faced by software teams during the late twentieth century [12]. Traditional development models, particularly the Waterfall approach, often resulted in rigid processes, lengthy delivery cycles and limited adaptability to changing requirements. This mismatch between business needs and delivery capabilities created what was widely referred to as the "software crisis." In response, practitioners and researchers sought alternative approaches that emphasized adaptability, collaboration and iterative progress [13]. These efforts culminated in the publication of the Agile Manifesto in 2001 which outlined values such as customer collaboration, responding to change, and delivering working software frequently. The manifesto did not prescribe a single method but rather established guiding principles that shaped a variety of frameworks. Among these, Extreme Programming (XP) and Scrum emerged as two of the most influential ones [14]. The evolution of software development methodologies is shown in Figure 1.

2.2 Emergence of Extreme Programming (XP)

Extreme Programming (XP) emerged in the late 1990s as a response to the growing need for high-quality software delivered at a rapid pace [15]. Introduced by Kent Beck during his work on the Chrysler Comprehensive Compensation System project, XP placed a strong emphasis on engineering practices that

directly improved code quality and team collaboration. Unlike other methodologies of its time, XP pushed traditional programming practices to their "extreme" which is how it derived its name. While XP gained popularity among smaller highly technical teams, it faced challenges in large-scale adoption due to its heavy reliance on disciplined engineering culture [16]. Nevertheless, many of its practices became foundational elements in modern software development and continue to influence frameworks used in recent years.

2.3 Emergence of Scrum

Scrum originated in the mid-1990s through the work of Jeff Sutherland and Ken Schwaber, who sought to address inefficiencies in traditional project management approaches [17]. Inspired by empirical process control and iterative development, Scrum emphasized short and time-boxed iterations known as sprints within which teams delivered potentially shippable product increments. The framework introduced well-defined roles including Product Owner, Scrum Master and Development Team along with ceremonies such as sprint planning, daily stand-ups, sprint reviews, and retrospectives [18]. This structure encouraged transparency and adaptation which enabled teams to respond more effectively to evolving customer needs. By the early 2000s, Scrum had become one of the most widely used agile frameworks. It established itself as the foundation for scaling models and serving as a gateway for many organizations transitioning into agile practices [19].

2.4 Evolution of Agile and the Role of Artificial Intelligence

Agile practices have undergone significant evolution since their formal introduction in 2001 [20]. During the 2010s, Agile shifted from being an experimental approach to becoming main stream. Enterprises, government agencies and large-scale projects began adopting Agile at scale supported by frameworks such as SAFe (Scaled Agile Framework) and LeSS (Large-Scale Scrum). At this stage, Agile was no longer seen as a fringe methodology but as a strategic enabler of organizational agility. The emphasis extended beyond software delivery to include culture, collaboration and responsiveness to market demands [21]. In recent years, Agile has further evolved to integrate with paradigms such as DevOps and continuous delivery. The evolution of agile has also been shaped by the growing influence of artificial intelligence and automation. AI is now embedded in core practices, enhancing decision-making, reducing repetitive tasks and enabling more predictive and adaptive processes [22]. Rather than replacing agile principles, artificial intelligence extends their applicability marking a shift toward more intelligent and data-informed frameworks that preserve agility while amplifying efficiency [23].

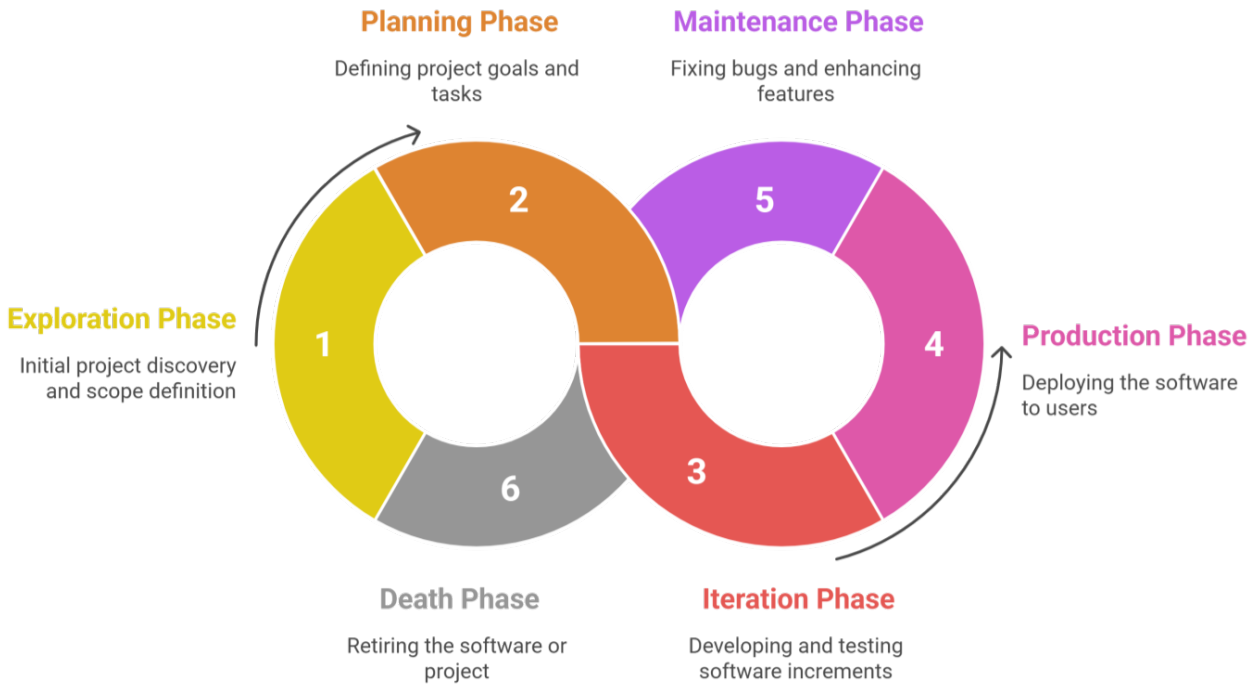


Figure 2. Phases of Extreme Programming.

3 Extreme Programming (XP)

Extreme Programming (XP) is an agile software development methodology designed to improve software quality and responsiveness to changing customer requirements [24]. XP emphasizes close collaboration between developers and customers, rapid feedback and disciplined engineering practices. Its purpose is to deliver high-value software through short frequent iterations while maintaining flexibility to adapt as requirements evolve. Unlike frameworks that focus primarily on project management, XP centers on technical excellence. It ensures that code remains tested and easily adaptable.

3.1 Phases of XP

The Extreme Programming lifecycle can be understood through six key phases. Each phase emphasizes iterative development and continuous collaboration. Figure 2 illustrates the XP lifecycle from exploration to the final death phase.

3.1.1 Exploration Phase

In this phase, customers and developers collaborate to explore project needs and expectations. Customers share initial requirements and possible features while developers experiment with technologies, tools, and architecture. The focus is on learning, discovery and building a shared understanding rather than finalizing details. Teams also begin writing user stories which act as building blocks for later iterations [25]. By the end of this phase, both the customer and development team have a clearer idea of the project scope and feasibility. This sets the foundation for a flexible and adaptive development cycle.

3.1.2 Planning Phase

The planning phase transforms exploration outcomes into a clear project direction. Customers and developers estimate time, costs and priorities based on user stories. XP uses the concept of the "planning game" where business stakeholders decide what to build and developers decide how much can be built within a given iteration. This collaborative process ensures realistic timelines and balanced workload distribution. The phase also helps in defining release plans and iteration schedules. It emphasizes communication and trust between customers and developers which keeps expectations aligned and goals achievable.

3.1.3 Iteration Phase

This phase involves short development cycles. These are usually one to three weeks long where small increments of the system are built and tested. Each iteration delivers working software that can be reviewed by the customer. Developers apply XP practices such as test-driven development, pair programming, continuous integration and refactoring during this phase [26]. Customer feedback at the end of each iteration guides future work and ensures alignment with real needs. Frequent releases reduce risks by catching problems early and maintaining steady progress. This phase highlights XP's emphasis on adaptability and continuous improvement.

3.1.4 Production Phase

Once the system is stable and meets core requirements, the focus shifts toward preparing it for wider release. Extra testing, performance tuning and fine-tuning of features are done to ensure quality. Documentation may be expanded where necessary to support deployment and future maintenance. Teams also assess technical debt and resolve issues that could affect stability [27]. This phase ensures that the product can perform reliably in real-world use. Customer input remains valuable, but the primary focus at this stage shifts toward quality assurance and ensuring the system is ready for deployment.

3.1.5 Maintenance Phase

After release, the system enters the maintenance phase where new requirements, bug fixes and changes continue to emerge. The team applies XP practices to keep the system adaptable and robust [28]. Continuous integration, automated testing and refactoring help in maintaining code quality while introducing changes. Customer feedback remains central, guiding which features or fixes are prioritized. This phase ensures that the system evolves with changing business needs without losing stability or performance. It demonstrates XP's long-term commitment to sustainable development.

3.1.6 Death Phase

The death phase occurs when the system no longer requires major changes or when it is replaced by a new solution. At this point, development slows down and the focus shifts to minor updates or eventual decommissioning. The team may write final documentation and hand over the system to operations or customers. This phase signals the end of active development but reflects that the project



Figure 3. Core values of XP.

was successfully delivered. The goal is to close the project smoothly while ensuring value is preserved for stakeholders.

3.2 Core Philosophy of XP

The core philosophy of Extreme Programming is built around five fundamental values that guide team behavior and development practices [29]. These values form the foundation for every decision made within an XP project shaping both technical approaches and interpersonal interactions. Figure 3 highlights the five values that form the foundation of XP. These values are:

1. Communication: Ensures that team members and stakeholders maintain constant dialogue to minimize misunderstandings and enhance collaboration.
2. Simplicity: Focuses on delivering the simplest solution that works, avoiding unnecessary complexity and making code easier to maintain.
3. Feedback: Encourages rapid learning and adaptation through continuous testing, pair programming, and frequent customer reviews.
4. Courage: Empowers developers to make necessary changes, refactor code or address defects without hesitation.
5. Respect: Fosters a collaborative environment where each team member's contributions are valued, creating trust and cohesion.

These values collectively enable XP teams to deliver high-quality software while remaining adaptable, efficient and closely aligned with customer needs.

3.3 Key Practices in XP

Extreme Programming emphasizes technical excellence and customer-centric development through structured yet adaptable practices [30]. By combining iterative development, close collaboration and continuous feedback, XP ensures that teams can respond effectively to changing requirements. These practices not only improve code quality but also foster a culture of shared responsibility and learning within the team. The key practices of XP are:

- Pair Programming: Two developers work together at a single workstation continuously reviewing each other's code to reduce errors and promote knowledge sharing.
- Test-Driven Development (TDD): Automated tests are written before coding to ensure that functionality meets requirements and supports safe refactoring.
- Continuous Integration: Code changes are integrated into a shared repository frequently, allowing early detection and resolution of conflicts.
- Refactoring: Code is regularly improved for simplicity, clarity, and maintainability, reducing technical debt over time.
- Small, Frequent Releases: Delivering working software in short iterations allows customers to see results quickly and provide timely feedback.
- Collective Code Ownership: All team members share responsibility for the code base. They encourage collaboration and accountability.
- Sustainable Pace: Teams maintain a work rhythm that avoids burnout, ensuring consistent

Table 1. Benefits and limitations of extreme programming (XP).

Aspect	Strengths	Limitations
Code Quality	Encourages high-quality code through TDD, continuous integration and refactoring	Requires disciplined practices; errors may occur if standards are not followed
Collaboration	Close teamwork via pair programming and collective ownership improves communication and knowledge sharing	Heavy reliance on collaboration; may be challenging for distributed or large teams
Customer Feedback	Frequent releases and customer involvement ensure alignment with user needs	Continuous customer availability is required, which may not always be feasible
Flexibility	Highly adaptable to changing requirements and priorities	Not easily scalable for very large or complex projects
Productivity	Short iterations and sustainable pace maintain steady progress	Teams unfamiliar with XP may experience a learning curve
Risk Management	Early detection of defects and risks through iterative testing and feedback	Relies on team discipline and experience for effective risk management
Knowledge Sharing	Promotes learning and skill transfer among team members through pair programming and collaboration	May slow down development initially if team members are inexperienced

productivity over the project lifecycle.

3.4 Advantages and Limitations

Extreme Programming offers a range of strengths that makes it highly effective for delivering high-quality software in an agile environment but it also has certain limitations that teams must consider [31]. Its emphasis on collaboration and disciplined engineering practices provides tangible benefits such as improved code quality, rapid feedback and enhanced adaptability. At the same time, successful implementation requires strong team discipline, customer involvement and familiarity with XP practices. Table 1 summarizes the key advantages and limitations of XP. It highlights the areas where it excels and potential challenges that organizations may encounter.

3.5 Applications of Extreme Programming

XP is highly applicable in software development environments that demand rapid delivery, continuous quality improvement and strong collaboration between developers and stakeholders [32]. It is particularly effective for small to medium-sized teams working on complex systems where requirements are expected to evolve frequently. The iterative nature of XP allows teams to respond quickly to changing business needs. It makes it ideal for web development, mobile applications and enterprise software where user expectations and technology trends can shift rapidly. XP is also widely adopted in startup environments and innovation-driven projects where early and continuous delivery of working software is critical to validate ideas and gather customer feedback. Additionally, XP practices have influenced modern DevOps pipelines, particularly in automated testing, continuous deployment and refactoring processes to help organizations reduce defects and accelerate release cycles. Beyond traditional software development, XP principles are applied in mission-critical systems such as healthcare software, financial platforms and e-commerce solutions where both quality and responsiveness are essential [33]. By fostering a culture of shared ownership and continuous improvement, XP enables teams to maintain a sustainable pace while delivering features that closely align with customer needs.

3.6 Modern Relevance of XP

Extreme Programming (XP) continues to play an important role in modern software development. Its value lies in technical practices that improve code quality and long-term sustainability. These practices remain critical even in AI-driven environments. Artificial intelligence can now support coding, testing, and defect detection, but XP ensures these tools are applied in a disciplined way. In DevOps pipelines, XP's focus on frequent releases and close customer feedback aligns naturally with continuous delivery models [34]. As teams become more distributed, practices such as pair programming and collective code ownership help

maintain knowledge sharing and code consistency. These qualities make XP useful for projects that require high technical rigor and flexibility. It is especially relevant in projects with rapidly changing requirements where adaptability must be combined with strong engineering discipline. XP also helps organizations balance speed with quality. While AI tools can automate repetitive tasks, they cannot replace human collaboration and decision-making. XP's practices encourage teamwork, communication, and shared responsibility which are essential for long-term project success. Instead of being replaced by modern technologies, XP provides the engineering discipline and human-centered practices that allow organizations to use new tools effectively while keeping software quality and agility intact. Its continued relevance shows that strong technical foundations remain just as important as new innovations in shaping successful software projects.

4 Scrum

Scrum is a widely adopted agile framework designed to help teams deliver high-quality software through iterative development and continuous collaboration [35]. Scrum provides a structured yet flexible approach that emphasizes transparency and adaptation. Unlike methodologies that focused primarily on engineering practices, Scrum centers on project management, defining roles, events, and artifacts that enable teams to work incrementally and respond effectively to changing requirements. Its lightweight framework has made it popular across industries from startups to large enterprises. It serves as the foundation for scaling models and hybrid agile approaches.

4.1 Scrum Framework and Phases

Scrum organizes work into iterative cycles called sprints that typically last for two to four weeks [36]. This allows teams to deliver incremental value regularly. The framework revolves around a set of clearly defined roles, artifacts and ceremonies that ensure continuous improvement. Figure 4 illustrates the Scrum workflow, moving from backlog refinement to sprint execution and review.

4.1.1 Product Backlog Refinement

The Product Backlog is a living list of features, enhancements, bug fixes and technical work that the team may deliver. It changes with business priorities and customer needs. It makes sure development stays aligned with organizational goals. In backlog refinement sessions, the team reviews items, estimates effort and clarifies requirements [37]. High-priority items are broken into smaller tasks that can be completed within a sprint. This process improves transparency and lets stakeholders see which features are planned. Refinement also helps the team spot dependencies, risks and technical challenges early which provide a clearer roadmap. By keeping the backlog well maintained, teams can maximize value delivery and remain flexible in handling changing requirements.

4.1.2 Sprint Planning

Sprint Planning marks the beginning of a sprint and sets the foundation for what the team will deliver [38]. During this meeting, the team selects items from the Product Backlog based on priority and team capacity to create a focused Sprint Backlog. Each item is analyzed and tasks are broken down to ensure clear understanding and achievable goals within the sprint. The team discusses dependencies, potential risks and approaches to complete each task effectively. Sprint Planning emphasizes collaboration between the Development Team and Product Owner to balance value delivery with realistic workload expectations. By the end of the session, the team commits to a clear sprint goal and is able to provide a shared vision for the upcoming iteration.

4.1.3 Sprint Execution

The Sprint Execution phase involves the actual development work to complete the items in the Sprint Backlog. Team members collaborate closely and apply cross-functional skills to deliver high quality increments of software. Practices such as code reviews, automated testing and continuous integration are often applied to maintain quality and reduce risks. Progress is tracked daily and any obstacles are immediately addressed to ensure the sprint remains on track [39]. Flexibility is maintained within the sprint to adapt to minor changes while preserving the sprint goal. Sprint Execution emphasizes both technical excellence and collaboration. This makes sure that work aligns with planned objectives. By the end of this phase, the team aims to deliver a fully functional and potentially shippable increment of the product.

4.1.4 Daily Scrum / Stand-ups

Daily Scrum meetings are short time-boxed gatherings designed to synchronize team activities and identify impediments. Each team member shares progress and plans for the day or any blockers that may hinder delivery. These stand-ups foster transparency

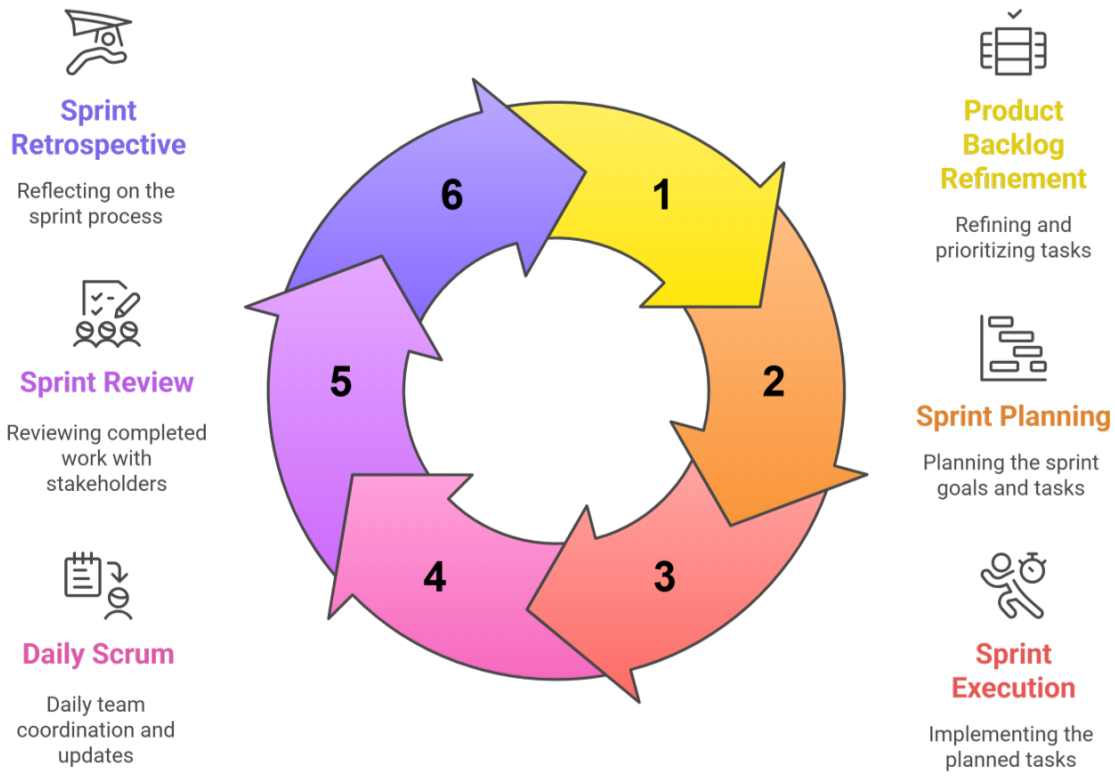


Figure 4. Phases of the Scrum Framework.

and team cohesion which helps the group remain focused on sprint goals. Impediments raised during the meeting are addressed immediately or escalated to ensure they do not affect the progress. Daily Scrums also provide a platform for collaborative problem-solving for allowing the team to adjust workflows and priorities in real-time. By maintaining consistent communication, teams can reduce misunderstandings and stay aligned with sprint objectives. This practice reinforces Scrum's principle of iterative inspection and adaptation while keeping development on track.

4.1.5 Sprint Review

At the end of each sprint, the Sprint Review provides an opportunity to inspect the delivered increment and gather feedback from stakeholders. The team demonstrates completed features, discusses what went well and identifies areas for improvement. Stakeholders provide input on priority adjustments, additional requirements or changes in scope [40]. This review ensures that the product evolves in alignment with business needs and user expectations. Sprint Reviews also promote transparency by making the team's work visible to stakeholders and encouraging collaborative decision-making. It serves as a checkpoint for validating assumptions and making informed decisions about future work. Feedback collected during this phase directly informs the next sprint, closing the loop for continuous improvement and value delivery.

4.1.6 Sprint Retrospective

The Sprint Retrospective focuses on process improvement, team dynamics and learning rather than product features. Retrospectives encourage open communication, psychological safety and a culture of continuous improvement. Lessons learned may include technical practices, collaboration strategies or workflow adjustments [41]. The goal is to enhance efficiency and remove impediments while maintaining team morale over the long term. Retrospectives also foster innovation by allowing the team to experiment with new approaches in subsequent sprints. This phase reinforces Scrum's commitment to inspect-and-adapt cycles, ensuring that teams not only deliver value but also evolve in their ways of working. By consistently applying insights from retrospectives, teams become more resilient and aligned with both project and organizational goals.

4.2 Core Philosophy of Scrum

Scrum is guided by a set of fundamental values and principles that shape how teams collaborate and make decisions to deliver software iteratively [42]. By internalizing these values, Scrum teams create

Table 2. Benefits and limitations of scrum.

Aspect	Strengths	Limitations
Flexibility	Highly adaptable to changing requirements and business priorities	Frequent changes may disrupt teams if not managed effectively
Collaboration	Promotes strong team interaction and stakeholder engagement	Heavy reliance on communication; distributed teams may face challenges
Transparency	Clear visibility of progress through artifacts and ceremonies	Requires disciplined tracking and active participation from all team members
Incremental Delivery	Delivers working software frequently, enabling early feedback and adjustments	May not suit projects needing a fixed, detailed plan upfront
Productivity	Time-boxed sprints and clear goals maintain steady progress	Teams new to Scrum may initially struggle with velocity estimation
Continuous Improvement	Retrospectives foster learning, process enhancements, and innovation	Requires commitment and openness; some teams may resist change
Accountability	Defined roles and responsibilities increase ownership and alignment	Misunderstanding of roles can lead to conflicts or inefficiencies
Stakeholder Engagement	Regular feedback ensures alignment with business and customer needs	Continuous involvement of stakeholders may be challenging in some contexts

an environment that encourages trust, collaboration and sustainable productivity. Figure 5 highlights the philosophy of Scrum. It can be summarized as follows:

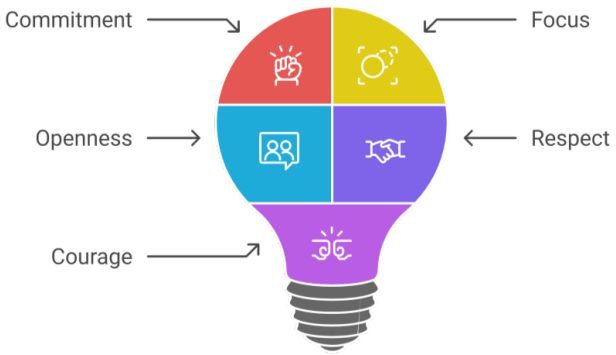


Figure 5. Core Values of Scrum.

- Commitment: Team members are dedicated to achieving sprint goals and delivering valuable outcomes consistently.
- Focus: Scrum ensures attention remains on the work planned for the sprint, minimizing distractions and enhancing productivity.
- Openness: It promotes transparency in progress, challenges and decisions, enabling informed collaboration among team members and stakeholders.
- Respect: It encourages valuing every individual's contributions, fostering trust, cohesion and a collaborative team environment.
- Courage: It empowers teams to embrace change, address challenges proactively, and experiment with innovative approaches to improve processes and deliverables.

These values form the backbone of Scrum's iterative framework. They guide teams in maintaining alignment with customer needs and continuously improving workflows. These values help in delivering high-quality software efficiently.

4.3 Practices in Scrum

Scrum relies on a set of disciplined practices that enable teams to deliver high-quality software in an iterative and collaborative manner [43]. The key practices of Scrum include:

- Time-Boxed Sprints: Development occurs in fixed-length iterations, typically 2–4 weeks, ensuring predictable delivery cycles and frequent feedback.
- Incremental Delivery: Each sprint produces a potentially shippable product increment, allowing stakeholders to inspect and provide feedback early and often.
- Cross-Functional Teams: Teams consist of members with complementary skills who collaborate to complete work without handoffs or silos.
- Scrum Roles: Clearly defined roles—Product Owner, Scrum Master, and Development

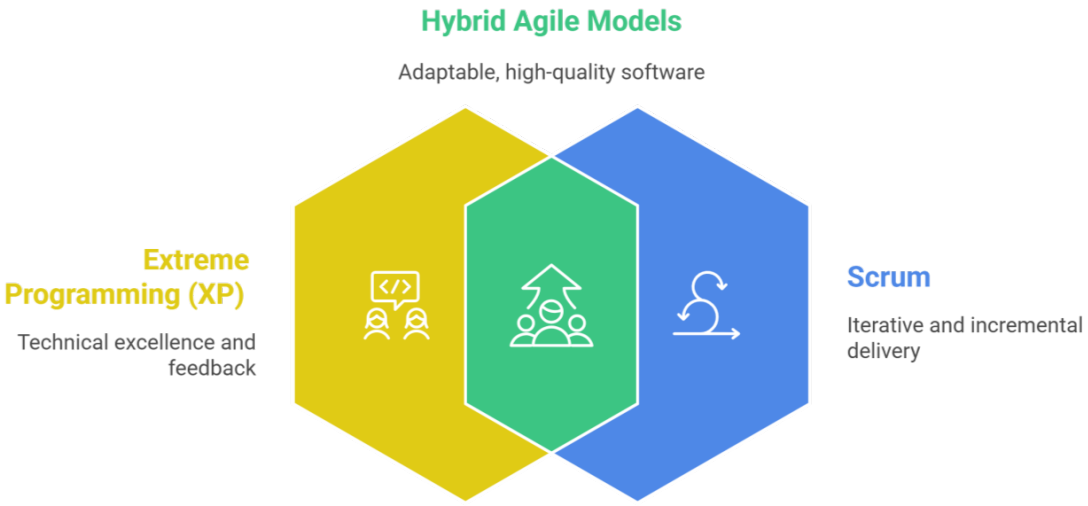


Figure 6. Hybrid Agile models combining Extreme Programming (XP) and Scrum.

Team—ensure accountability and structured collaboration.

- Scrum Ceremonies: Regular events such as Sprint Planning, Daily Scrum, Sprint Review, and Sprint Retrospective support planning, coordination, and continuous improvement.
- Product and Sprint Backlogs: Artifacts that provide transparency, prioritize work, and maintain alignment with evolving customer needs.
- Definition of Done (DoD): Establishes clear criteria for when work is considered complete, ensuring quality and consistency.

4.4 Advantages and Limitations

Scrum provides numerous benefits in agile software development, but it also has some limitations that teams must consider. Table 2 summarizes the key strengths and potential challenges of Scrum.

4.5 Applications of Scrum

Scrum is widely applied across industries and project types particularly where flexibility, iterative delivery, and stakeholder collaboration are essential. It is commonly used in software development projects including web and mobile applications, enterprise solutions and product development initiatives where requirements frequently change and incremental delivery adds value. Scrum is also popular in startups and innovation-driven environments where rapid prototyping and continuous feedback are critical for validating ideas and adapting to market demands [44]. Beyond software development, Scrum principles have been applied to non-IT domains, such as marketing, education, and operational process improvements, demonstrating its versatility. Large organizations often implement scaled Scrum frameworks like SAFe, LeSS, or Nexus to coordinate multiple teams working on complex projects, maintaining alignment and delivering integrated increments. With the adoption of DevOps practices, cloud platforms, and AI-powered project management tools, Scrum teams can automate workflows, track progress more effectively and enhance collaboration across distributed environments. Overall, Scrum's structured yet adaptive approach makes it a practical and effective framework for managing projects that demand both speed and quality in dynamic, fast-changing environments [45].

4.6 Modern Relevance of Scrum

Scrum remains highly relevant in modern software development. It provides structure through clear roles, ceremonies and time-boxed sprints. These elements provide clarity and rhythm to the teams. This is essential even with advanced tools and automation. Artificial intelligence can help with backlog prioritization, sprint forecasting, and progress tracking. However, Scrum ensures that these insights are used in a disciplined way. Its iterative approach encourages continuous inspection and adaptation. This makes Scrum effective in environments where requirements change quickly [46]. In DevOps pipelines, Scrum works alongside automation to keep business goals aligned with technical delivery. It ensures that every sprint focuses on customer value. For distributed and hybrid teams, Scrum ceremonies such as daily stand-ups and sprint reviews maintain transparency and accountability. These practices

Table 3. Benefits and limitations of hybrid models.

Aspect	Strengths	Limitations
Balance of Practices	Combines Scrum's structured roles with XP's technical rigor for a well-rounded approach	Risk of overlap or confusion when blending roles and practices
Delivery	Predictable sprint cycles ensure visibility while XP practices maintain code quality	Coordination overhead may slow progress if practices are not well integrated
Collaboration	Strong stakeholder engagement (Scrum) plus deep developer collaboration (XP)	Requires high communication discipline; harder with distributed teams
Technical Quality	Continuous integration, TDD, and refactoring reduce defects and improve maintainability	Demands skilled developers and consistent adherence to engineering practices
Flexibility	Adapts to changing requirements while maintaining quality and visibility	Complexity increases when scaling hybrid models across large organizations
Learning Curve	Builds a culture of shared ownership and continuous improvement	Teams need additional training to master both Scrum and XP simultaneously

create a shared rhythm that AI tools cannot replace. Scrum has adapted by integrating digital assistants, smart dashboards, and AI-powered analytics. Its strength lies in balancing technological support with human collaboration. It provides teams a clear process that guides decision-making, improves communication, and sustains delivery of customer value in fast-changing environments. Scrum also encourages a culture of learning and knowledge sharing. Teams continuously improve their skills and processes while delivering meaningful results.

5 XP and Scrum - Hybrid Models

Many organizations today adopt hybrid approaches that bring together the strengths of Extreme Programming (XP) and Scrum. XP contributes engineering discipline through practices such as test-driven development and continuous integration. Scrum, in contrast, provides well-defined roles and ceremonies that improve project visibility and stakeholder engagement. Using only one framework often leaves gaps. Scrum without technical practices may lead to technical debt. XP without structured management may struggle to scale in larger projects [47]. Hybrid models address these limitations by combining XP's technical rigor with Scrum's organizational structure. Figure 6 shows how XP and Scrum overlap to form hybrid approaches that balance engineering rigor with structured delivery.

This hybrid approach is becoming increasingly important in modern contexts such as distributed development, AI-driven automation, and high-complexity projects. In such environments, teams need both strong engineering quality and predictable delivery. Core Hybrid Practices commonly adopted by teams include:

- Scrum Sprints with XP Practices: Teams run fixed-length Scrum sprints but apply XP techniques like pair programming, refactoring, and TDD during sprint execution.
- Defined Scrum Roles with Technical Responsibilities: Scrum roles such as Scrum Master, Product Owner, and Development Team remain intact, while developers consistently use XP engineering practices.
- Scrum Ceremonies Enhanced with XP Feedback: Sprint planning, reviews, and retrospectives guide delivery. These are complemented by XP's continuous collaboration with customers and rapid feedback loops.
- Backlog Refinement with Technical Priorities: The Product Backlog manages user stories but also includes technical tasks such as automated testing and code refactoring drawn from XP.
- Integration of CI/CD Pipelines: Continuous integration and frequent releases, which are central to XP, are combined with Scrum's incremental delivery framework to improve speed and reliability.

By blending both methods, hybrid Scrum–XP models create a balanced way of working. Teams benefit from Scrum's structure and visibility while also maintaining the high code quality that XP promotes. The challenge lies in managing overlaps and avoiding role confusion.

Teams may also need additional training to handle both frameworks effectively. When applied carefully, however, hybrid models offer a strong pathway for organizations that want both structured project management and sustainable engineering excellence. Table 3 highlights the key benefits and limitations of hybrid models.

6 Comparative Analysis of XP, Scrum and Hybrid Models

Extreme Programming (XP), Scrum, and their hybrid combinations represent three of the most widely applied approaches in agile software development. Each offers distinct strengths and addresses different challenges but they also share a foundation in agile principles. XP emphasizes technical excellence and disciplined engineering practices. Scrum focuses on project management, structured roles, and iterative delivery. Hybrid models aim to merge these two dimensions by combining Scrum's organizational discipline with XP's technical rigor. Understanding how these approaches align, differ, and complement each other is essential for practitioners seeking to select or adapt frameworks for their projects. This section therefore provides a comparative analysis that highlights similarities, key differences and the rationale for adopting hybrid approaches in modern software development.

6.1 Similarities across XP, Scrum, and Hybrid Models

XP, Scrum, and their hybrid combinations are all rooted in the core principles of the Agile Manifesto. Each approach emphasizes adaptability, collaboration, and delivery of customer value through iterative development. Table 4 presents similarities among them.

6.2 Key Differences

Although XP, Scrum, and their hybrid models share a foundation in agile principles, they differ in focus, structure, and implementation. XP emphasizes technical excellence through disciplined engineering practices. Scrum prioritizes structured project management with defined roles and ceremonies. Hybrid models attempt to merge these strengths but introduce new challenges of integration, role clarity, and coordination overhead. Understanding these differences helps practitioners select the most suitable framework for their project environment. Table 5 presents the key differences between XP, Scrum and their Hybrids.

7 Decision Framework for Practitioners

In this section, we proposed a practical decision framework developed from the comparative analysis of XP, Scrum, and hybrid models. While existing studies often stop at describing similarities and differences, our contribution goes further by translating these insights into actionable guidance. The framework is designed to help practitioners systematically decide whether to adopt XP, Scrum, or a hybrid approach based on their project context, team characteristics, and stakeholder needs. It provides criteria for evaluation, step-by-step guidance for selection and conceptual use cases that demonstrate how the framework can be applied in real-world scenarios. We also validated our framework by providing some real world previously published case studies from recent years. By presenting this framework, we aim to bridge the gap between theoretical comparisons and practical adoption, giving teams a structured tool to make informed and context-sensitive choices.

7.1 Core Evaluation Criteria

Our proposed decision framework is grounded in a set of key criteria that determine the suitability of XP, Scrum, or hybrid approaches. This criterion is presented in Figure 7. These criteria capture the most important project and team characteristics that influence the effectiveness of an agile methodology. By assessing these factors, practitioners can align methodology choice with their actual context rather than relying on generic recommendations. The following points outline the core criteria that form the foundation of our framework.

- Team Size and Composition – whether the team is small and highly technical or larger and distributed.
- Project Complexity and Domain – the level of technical challenges and management needs.
- Customer Involvement – how frequently stakeholders are available for collaboration.
- Requirement Stability – whether requirements change frequently or remain relatively stable.
- Technical Quality Priorities – the importance of practices such as TDD, refactoring, and continuous integration.
- Risk Tolerance – how much uncertainty and experimentation the project can accommodate.

Table 4. Key similarities of Extreme Programming (XP), Scrum and Hybrid models.

Similarity Aspect	Description
Iterative Development	All models rely on short cycles (XP iterations, Scrum sprints, Hybrid combines both).
Customer Collaboration	Each framework emphasizes active customer or stakeholder involvement throughout the project.
Adaptability to Change	All are designed to respond effectively to evolving requirements and shifting priorities.
Team Collaboration	Cross-functional teamwork, knowledge sharing, and shared ownership are central to all approaches.
Focus on Quality	Every model prioritizes high-quality, functional software delivered at each cycle.
Transparency & Visibility	Progress, blockers, and priorities are made visible through ceremonies, artifacts, or frequent releases.
Feedback Loops	Regular reviews, retrospectives, and testing cycles ensure fast learning and adjustments.
Continuous Learning	Teams are encouraged to reflect, experiment, and refine practices to improve performance.
Incremental Value Delivery	Each iteration produces working software that provides measurable value to customers.
Sustainable Pace	All promote a balanced workload that avoids burnout and supports steady delivery.
Empowered Teams	Teams are self-organizing and trusted to make decisions about implementation and delivery.
Risk Reduction	Short cycles, regular testing, and early feedback help reduce risks and detect issues quickly.
Alignment with Agile Values	All approaches reflect the Agile Manifesto values of collaboration, adaptability, and customer focus.
Applicability Beyond Software	Each model has influenced practices outside IT, showing flexibility in broader organizational contexts.

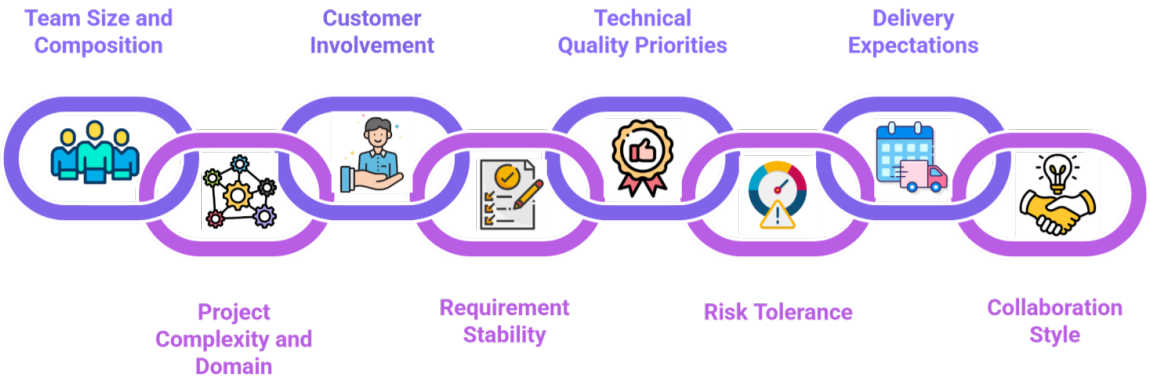


Figure 7. Evaluation Criteria of our Framework.

- Delivery Expectations – the need for predictable timelines versus flexible adaptation.
- Collaboration Style – whether the team thrives on structured ceremonies or intensive peer collaboration.

Together, these criteria provide the foundation for applying the framework in practice and serve as the basis for the step-by-step selection process discussed in the next subsection.

7.2 Steps for Applying the Decision Framework

The evaluation criteria formed the foundation of our decision framework. To make these criteria actionable,

Table 5. Key Differences between Extreme Programming (XP), Scrum and Hybrid models.

Aspect	Extreme Programming (XP)	Scrum	Hybrid (Scrum–XP)
Primary Focus	Engineering excellence and code quality	Project management, roles, and iterative delivery	Balance of structured management and disciplined engineering
Roles	Minimal roles (team and customer)	Defined roles: Product Owner, Scrum Master, Development Team	Scrum roles retained, with added emphasis on XP technical practices
Iteration Style	Flexible iterations based on user stories	Fixed-length sprints (2–4 weeks)	Scrum sprints enriched with XP's technical activities
Ceremonies / Events	Informal meetings and reviews	Structured ceremonies: planning, stand-ups, reviews, retrospectives	Scrum ceremonies with additional XP collaboration and feedback
Technical Practices	Strong focus: TDD, pair programming, CI, refactoring	Minimal guidance; technical practices chosen by the team	XP's practices integrated into Scrum's sprint cycle
Documentation	Minimal; working software is primary measure	Artifacts such as Product and Sprint Backlogs	Scrum artifacts plus explicit inclusion of technical backlog items
Customer Involvement	Continuous, hands-on involvement	Engagement mainly during refinement, planning, and reviews	Combines Scrum's structured touchpoints with XP's frequent collaboration
Feedback Frequency	Continuous (tests, reviews, pair programming)	At the end of each sprint via reviews/retrospectives	Continuous feedback from XP plus sprint-based feedback loops
Scalability	Best for small, highly technical teams	Scales more easily with frameworks like SAFe or LeSS	More complex to scale due to blending roles and practices
Risk Management	Managed through continuous testing and iteration	Managed through sprint reviews and artifacts	Combines both but may add overhead if integration is weak
Flexibility	Highly adaptable to changing technical requirements	Adaptable but constrained within sprint boundaries	Balanced flexibility with structure, though integration challenges may limit agility
Suitability	High-risk, technically complex, small to medium teams	Larger teams and projects needing structure and visibility	Distributed, complex, or high-stakes projects requiring both quality and predictability

we propose a step-by-step process that guides practitioners from evaluation to final methodology selection. These steps translate abstract considerations into practical actions ensuring that the choice of XP, Scrum, or a hybrid model is systematic, transparent and aligned with project needs. Figure 8 sums up these steps.

Step 1: Assess Team Characteristics.
Evaluate the team's size, skills and distribution. XP is more suitable for small to medium-sized teams with strong technical expertise and close collaboration. Scrum accommodates larger or distributed teams through its structured roles and ceremonies. When teams are cross-functional but also highly technical, a hybrid approach ensures both coordination and quality. This step ensures that the selected framework matches the team's natural strengths and avoids overburdening them with practices they cannot sustain.

Step 2: Evaluate Project Complexity.
Analyze the level of technical challenges and the project domain. Projects involving complex algorithms, high-quality standards, or innovation often require XP's strong engineering practices. Scrum works better when projects demand predictable delivery cycles. It also supports coordination across multiple stakeholders and clear accountability. Hybrid approaches can serve projects that combine technical innovation with organizational complexity which allow both structured delivery and technical rigor.

Step 3: Determine Customer Involvement Needs.

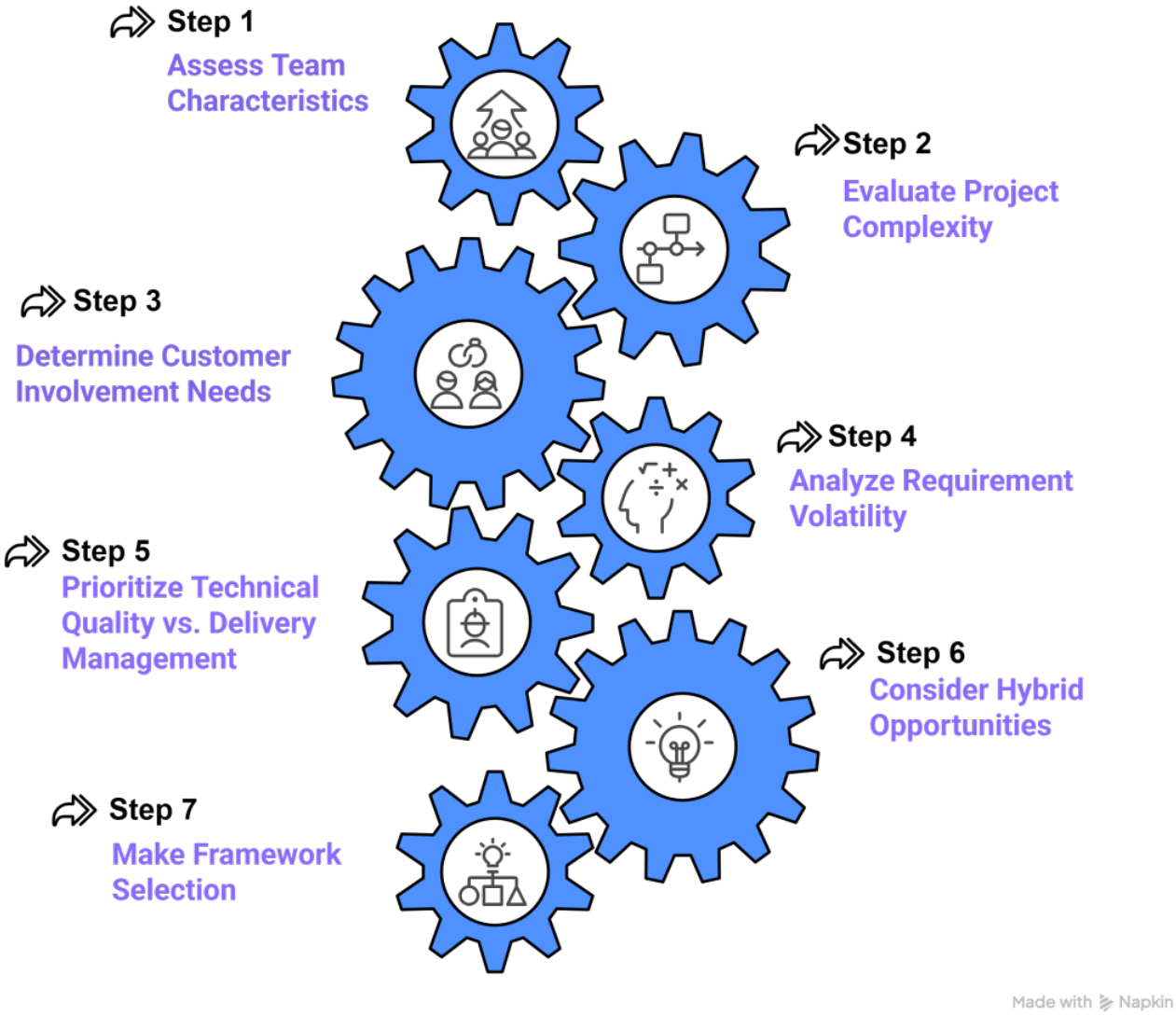


Figure 8. Steps for the Proposed Decision Framework.

Consider how often stakeholders or end users can realistically engage. XP assumes continuous customer presence which may not always be feasible in corporate environments. Scrum provides defined opportunities for involvement during backlog refinement, sprint planning, reviews and retrospectives. Hybrids allow flexibility by maintaining Scrum's structured ceremonies while encouraging more frequent feedback loops inspired by XP. This step ensures expectations for stakeholder collaboration are realistic and achievable.

Step 4: Analyze Requirement Volatility.
Examine how stable the project requirements are. XP adapts quickly to changing requirements with practices such as refactoring and continuous feedback. Scrum allows for controlled adaptation within sprint boundaries but avoids major changes mid-sprint. Hybrid models can offer balance and enable teams to stay flexible without losing focus on sprint goals. Correctly gauging volatility ensures that the methodology provides responsiveness without sacrificing delivery stability.

Step 5: Prioritize Technical Quality vs. Delivery Management
Clarify whether the primary concern is software craftsmanship or structured delivery. XP enforces technical excellence through practices like TDD, pair programming, and continuous integration. Scrum emphasizes transparency, planning, and predictable delivery but leaves technical practices up to the team. Hybrids combine both by embedding XP's engineering discipline within Scrum's delivery cycles. This step aligns the chosen framework with the organization's definition of "success," whether that means code quality, stakeholder satisfaction, or both.

Step 6: Consider Hybrid Opportunities
Identify situations where blending practices may provide additional value. For example, Scrum's sprint reviews can ensure visibility to stakeholders while XP's pair programming and automated testing maintain quality throughout development. In distributed teams, Scrum's structured coordination can be combined with XP's technical practices to overcome distance-related challenges. This step prevents teams from treating XP and Scrum as mutually exclusive and encourages a tailored approach.

Step 7: Make Framework Selection
Synthesize the insights from the previous steps to make a deliberate choice. If technical excellence and adaptability are critical, XP may be best. If structured coordination and scalability are priorities, Scrum is more suitable. When both dimensions are equally important, a hybrid approach ensures balance. This step formalizes the decision and provides clarity to the team. It ensures that the methodology is not chosen by habit or trend but by alignment with actual needs.

7.3 Conceptual Use Cases

To illustrate how the proposed decision framework can be applied in practice, we present conceptual use cases. These are hypothetical but realistic scenarios that demonstrate how different project contexts map to XP, Scrum, or hybrid approaches. Each use case highlights the relevant criteria, the framework's recommendation. It also explains the reasoning behind the selection.

7.3.1 Use Case 1: Startup with Rapidly Changing Requirements (XP)

Context: A small startup team of six developers is building an innovative product where customer requirements evolve weekly. The team is technically strong and works closely with end users.

Framework Recommendation: Extreme Programming (XP).

Rationale: The small team size, high technical skill and rapidly changing requirements favor XP's practices such as TDD, pair programming, and continuous feedback.

7.3.2 Use Case 2: Large Enterprise Project with Multiple Stakeholders (Scrum)

Context: A financial services company is developing a compliance reporting tool with 40 developers spread across three locations. The project requires coordination with multiple departments and predictable delivery cycles.

Framework Recommendation: Scrum.

Rationale: Scrum provides defined roles, ceremonies and artifacts. It helps manage complexity and ensure visibility, and align multiple stakeholders on priorities.

7.3.3 Use Case 3: Distributed Product Development with both Technical and Organizational Demands (Hybrid)

Context: A multinational company is developing an AI-based healthcare application. The team is distributed across two countries combining data scientists and software engineers. Stakeholders demand both technical quality and reliable progress updates.

Framework Recommendation: Hybrid (Scrum-XP).

Rationale: The hybrid approach allows the team to maintain technical rigor (TDD, continuous integration, refactoring) while benefiting from Scrum's structured ceremonies, sprint planning and stakeholder visibility.

7.4 Real-world Case Studies

To validate our decision framework, we examined published case studies from recent years that report on the use of XP, Scrum, or hybrid models in practice. These case studies provide evidence that the criteria and steps we propose map effectively onto real-world project contexts. Each case demonstrates how methodology choices reflected the same considerations outlined in our framework.

7.4.1 Case Study 1: Hybrid Scrum-XP Model in Software Companies

Bose et al. [49] proposed a Hybrid Scrum-XP model to enhance the effectiveness of agile methodologies in Bangladeshi software companies. The study combined the strengths of Scrum's structured management practices with XP's focus on engineering techniques to address the limitations of both methods. A survey conducted among leading software companies in Bangladesh revealed a preference for agile methodologies with a significant inclination towards customizing models to suit project-specific needs. The proposed hybrid model received positive feedback indicating its potential to improve time and cost efficiency in software development. The study

suggests that integrating Scrum and XP can overcome their individual drawbacks and lead to more effective agile practices in the Bangladeshi context. The positive reception and preference for customization directly validate our framework's emphasis on hybrid opportunities when both technical quality and structured delivery are critical.

7.4.2 *Case Study 2: Agile Scrum for Web-based School Information System*

Wandri et al. [50] applied Scrum to develop a web-based school information system that improved administrative efficiency in vocational high schools. The study addressed inefficiencies caused by semi-manual processes using Word and Excel which were prone to errors. By employing Scrum, the project established clear roles (Scrum Master, Product Owner, and Team) and structured rituals (Sprint Planning, Daily Standups, Reviews, and Retrospectives). These ensured iterative progress and continuous stakeholder feedback. The system integrated features for student, teacher, staff, and financial management, along with interactive dashboards and real-time analytics. Evaluation showed a 95% user satisfaction rate and notable improvements in efficiency. This validates our framework's recommendation of Scrum for projects requiring clear accountability, predictable delivery and coordination across multiple stakeholders.

7.4.3 *Case Study 3: Enhancing XP Adoption through SAMM*

Abrar et al. [51] addressed the challenges organizations face in adopting Extreme Programming (XP) by developing a Scalable Agile Maturity Assessment Model (SAMAM). The model is based on 14 critical barriers to XP adoption identified through a Systematic Literature Review (SLR) and validated via an industry survey. SAMAM structures maturity into five levels, replacing traditional Key Process Areas (KPAs) with practices aimed at overcoming these barriers. Its effectiveness was evaluated through industrial case studies using the Motorola Assessment Tool. The results demonstrate that SAMAM provides a practical, scalable, and comprehensive approach for organizations to assess and improve their XP adoption maturity. This study created a maturity model to address barriers in adopting XP. It confirms our framework's view that XP works best with technically strong, smaller teams, but needs adaptation in larger or complex settings.

7.4.4 *Case Study 4: Scrum-XP Hybrid Methodology in Distributed Web Development*

Mustafa et al. [52] explored the application of a Scrum-XP hybrid methodology in developing a Clinic Appointment Booking Application (CABA) by distributed teams. The study demonstrates how combining Scrum's project management practices with XP's engineering practices improves communication, project coordination along with software quality. The hybrid approach resulted in enhanced resource management, product stability and higher customer satisfaction. Additionally, the study highlights how iterative feedback loops and collaborative coding practices from XP complement Scrum's structured sprints making the hybrid approach highly effective for geographically distributed teams. The study validates our framework's recommendation to adopt hybrid approaches when both technical rigor and structured delivery are needed.

7.4.5 *Case Study 5: Hybrid Scrum and Six Sigma in Software Development*

Alam et al. [48] explored the integration of Scrum and Six Sigma methodologies in software development through a practical case study. The hybrid approach aimed to leverage Scrum's agile project management with Six Sigma's focus on process improvement and quality control. The results indicated that while the combined model enhanced team performance and customer satisfaction, it also led to increased rework and a higher number of defects discovered. These outcomes were deemed acceptable as they aligned with the objectives of both Scrum and Six Sigma emphasizing continuous improvement and quality enhancement in the development process. Although not centered on XP, it validates our framework's broader claim that hybridization is an effective strategy when teams need to integrate agile flexibility with additional quality or process disciplines.

8 Future Direction in Agile Frameworks

As agile methodologies continue to evolve, there is a growing need to explore new directions that enhance their effectiveness and adaptability. Emerging technologies, changing team dynamics and increasing project complexity are shaping the way XP, Scrum and hybrid agile frameworks are applied in practice. Future research and industry initiatives focus on improving automation, scalability and collaboration while ensuring high-quality software delivery. The following points highlight the key trends and areas of innovation that are likely to influence the next

generation of agile practices.

AI and Automation in Agile: Incorporating artificial intelligence and machine learning for predictive sprint planning, automated code reviews, bug detection and quality assurance to enhance productivity and decision-making.

Hybrid and Scaled Agile Models: Further development of hybrid models (e.g., Scrum-XP, Scrum-Kanban, SAFe) to address scalability challenges in large, distributed, or complex projects.

Remote and Distributed Team Optimization: Researching tools, processes, and communication strategies to maximize collaboration and efficiency in geographically dispersed teams.

Continuous Integration of DevOps Practices: Deepening the integration of DevOps principles with agile frameworks to ensure faster delivery, higher reliability, and seamless alignment between development and operations.

Adaptive Learning and Agile Training: Developing AI-driven training programs and adaptive learning platforms to upskill teams rapidly for effective adoption of XP, Scrum, and hybrid methodologies.

Sustainability and Ethical Software Development: Promoting sustainable, socially responsible, and ethically aligned software development practices within agile frameworks.

9 Conclusion

Agile software development frameworks such as XP, Scrum and hybrid combinations remain central to modern software engineering, yet organizations often struggle to select or adapt the right approach. This study addressed that challenge through a dual contribution. First, we conducted a comparative analysis of XP, Scrum and hybrid models, highlighting their similarities, differences, strengths, and limitations. This analysis provided a clearer understanding of how each framework operates and where they are best applied. Second, building on this analysis, we proposed a decision framework that translates theoretical insights into actionable guidance. The framework identifies key criteria, outlines a step-by-step process for methodology selection and demonstrates practical application through conceptual use cases. To further validate its relevance, we examined published real-world case studies which showed strong alignment between our framework's recommendations and actual project practices. Together, the comparative analysis and decision framework contribute both to research and practice. The analysis sharpens understanding of agile methodologies while the framework offers practitioners a structured tool for context-sensitive adoption. For researchers, this work highlights opportunities for deeper empirical studies of hybrid models, large-scale applications, and integration with emerging practices such as DevOps and AI. For practitioners, it provides clarity and structure in navigating agile choices. Ultimately, the paper underscores the importance of both rigorous comparison and practical decision support in advancing the effectiveness of agile software development.

Data Availability Statement

Data will be made available on request.

Funding

This work was supported without any funding.

Conflicts of Interest

Shabib Aftab served as an Editor-in-Chief of the *ICCK Journal of Software Engineering* at the time of manuscript submission. To ensure the integrity of the peer-review process, Shabib Aftab was not involved in the editorial handling, peer review, or decision-making process for this manuscript, which was handled independently by another editor. The remaining authors declare no conflicts of interest.

AI Use Statement

The authors declare that no generative AI was used in the preparation of this manuscript.

Ethical Approval and Consent to Participate

Not applicable.

References

[1] Herdika, H. R., & Budiardjo, E. K. (2020, September). Variability and commonality requirement specification on agile software development: Scrum, xp, lean, and kanban. In *2020 3rd International Conference on Computer and Informatics Engineering (IC2IE)* (pp. 323–329). IEEE. [CrossRef]

[2] Akhtar, A., Bakhtawar, B., & Akhtar, S. (2022). Extreme programming vs scrum: A comparison of agile models. *International Journal of Technology Innovation and Management (IJTIM), 2*(2), 80-96. [CrossRef]

[3] Cohn, M. (2021). *User stories: For agile software development with Scrum, XP, and others*. BoD–Books on Demand.

[4] Salazar-Salazar, G., Mora, M., Duran-Limon, H., Alvarez-Rodriguez, F., & Munoz-Zavala, A. (2024). Review of Agile SDLC for Big Data Analytics Systems in the Context of Small Organizations Using Scrum-XP. *International Arab Journal of Information Technology (IAJIT), 21*(6). [CrossRef]

[5] Hosseini, S. (2023). Xcrum: A Synergistic Approach Integrating Extreme Programming with Scrum. *arXiv preprint arXiv:2310.03248*.

[6] Fitzgerald, B., Stol, K. J., O'Sullivan, R., & O'Brien, D. (2013, May). Scaling agile methods to regulated environments: An industry case study. In *2013 35th International Conference on Software Engineering (ICSE)* (pp. 863-872). IEEE. [CrossRef]

[7] Soukaina, M., Badr, E., Abdelaziz, M., & Nawal, S. (2021). Towards a new metamodel approach of Scrum, XP and Ignite methods. *International Journal of Advanced Computer Science and Applications, 12*(12), 192–202. [CrossRef]

[8] Mamun, M. N. H. (2024). Integration Of Artificial Intelligence And DevOps In Scalable And Agile Product Development: A Systematic Literature Review On Frameworks. *ASRC Procedia: Global Perspectives in Science and Scholarship, 4*(1), 01-32. [CrossRef]

[9] Reiter, M. (2025). Comparative Analysis of Agile Frameworks: Scrum, Kanban, Extreme Programming. In *Data-Centric Business and Applications: Advancing Success Through Operational Excellence, Financial Innovation, Digital Transformation, and Data-Driven Human Resource Management* (pp. 335-348). Cham: Springer Nature Switzerland. [CrossRef]

[10] Mora, M., Adelakun, O., Galvan-Cruz, S., & Wang, F. (2022). Impacts of IDEF0-based models on the usefulness, learning, and value metrics of Scrum and XP project management guides. *Engineering Management Journal, 34*(4), 574-590. [CrossRef]

[11] Magistretti, S., & Trabucchi, D. (2025). Agile-as-a-tool and agile-as-a-culture: a comprehensive review of agile approaches adopting contingency and configuration theories. *Review of Managerial Science, 19*(1), 223-253. [CrossRef]

[12] Whiteley, A., Pollack, J., & Matous, P. (2021). The origins of agile and iterative methods. *The Journal of Modern Project Management, 8*(3). [CrossRef]

[13] Owen, H., & Dunham, N. (2015). Reflections on the use of iterative, agile and collaborative approaches for blended flipped learning development. *Education Sciences, 5*(2), 85-103. [CrossRef]

[14] Shrivastava, A., Jaggi, I., Katoch, N., Gupta, D., & Gupta, S. (2021, July). A systematic review on extreme programming. In *Journal of Physics: Conference Series* (Vol. 1969, No. 1, p. 012046). IOP Publishing. [CrossRef]

[15] Pandit, D. P., Gajjam, N. S., Sangu, V. S., Chandra, S., & Chandrasatheesh, C. (2025). Optimizing Efficiency and Delivering Quality for Lean and Extreme Programming (XP) in Agile Business Methodologies. In *Impact of Digital Transformation on Business Growth and Performance* (pp. 547-578). IGI Global Scientific Publishing. [CrossRef]

[16] Wiratama, J., & Santoso, H. (2023). Developing a class scheduling mobile application for private campus in Tangerang with the Extreme Programming (XP) model. *G-Tech: Jurnal Teknologi Terapan, 7*(2), 484–493. [CrossRef]

[17] Beedle, M., Devos, M., Sharon, Y., Schwaber, K., & Sutherland, J. (1999). SCRUM: An extension pattern language for hyperproductive software development. *Pattern languages of program design, 4*(1), 637-651.

[18] Sassa, A. C., de Almeida, I. A., Pereira, T. N. F., & de Oliveira, M. S. (2023). Scrum: A systematic literature review. *International Journal of Advanced Computer Science and Applications, 14*(4). [CrossRef]

[19] Kalenda, M., Hyna, P., & Rossi, B. (2018). Scaling agile in large organizations: Practices, challenges, and success factors. *Journal of Software: Evolution and Process, 30*(10), e1954. [CrossRef]

[20] Bhattacharya, S., Shukla, K., & Shukla, A. (2024). Beyond Scrum: Anticipating the evolution of agile project management practices. In *Practical approaches to agile project management* (pp. 119–141). IGI Global. [CrossRef]

[21] Søderberg, A. M., Krishna, S., & Bjørn, P. (2013). Global software development: commitment, trust and cultural sensitivity in strategic partnerships. *Journal of International Management, 19*(4), 347-361. [CrossRef]

[22] Almashhadani, M., Mishra, A., & Yazici, A. (2024). Software maintenance practices using agile methods towards cloud environment: A systematic mapping. *Journal of Software: Evolution and Process, 36*(11), e2698. [CrossRef]

[23] Sarkar, T., Moharana, B., Rakhra, M., & Cheema, G. S. (2024, March). Comparative Analysis of Empirical Research on Agile Software Development Approaches. In *2024 11th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions)(ICRITO)* (pp. 1-6). IEEE. [CrossRef]

[24] Tabassum, A., Manzoor, I., Bhatti, S. N., Asghar, A. R., & Alam, I. (2017). Optimized quality model for agile development: extreme programming (XP) as a case scenario. *International Journal of Advanced Computer Science and Applications, 8*(4).

[25] Sami, M. A., Rasheed, Z., Waseem, M., Zhang, Z.,

Herda, T., & Abrahamsson, P. (2024). Prioritizing software requirements using large language models. *arXiv preprint arXiv:2405.01564*.

[26] Nayaka Sheetakallu Krishnaiah, P., Narayan, D. L., & Sutradhar, K. (2024). A survey on secure metadata of agile software development process using blockchain technology. *Security and Privacy, 7*(2), e342. [CrossRef]

[27] Eldanasory, N. A., Idrees, A. M., & Yehia, E. (2024). EFSP: An enhanced full Scrum process model. *International Journal of Software Engineering and Knowledge Engineering, 34*(5), 729–749. [CrossRef]

[28] Zahedi, M. H., Kashanaki, A. R., & Farahani, E. (2023). Risk management framework in Agile software development methodology. *International Journal of Electrical & Computer Engineering* (2088-8708), 13(4). [CrossRef]

[29] Ahmed, I., Munir, F., Nafees, F., Mahmood, J., Chaudhry, S. A., Javaid, M., & Batool, K. (2024, December). The influence of Kanban agile methodology on software project management: A survey method. In *2024 International Conference on Engineering and Emerging Technologies* (*ICEET*) (pp. 1–6). IEEE. [CrossRef]

[30] Madampe, K., Hoda, R., & Grundy, J. (2024). Supporting emotional intelligence, productivity and team goals while handling software requirements changes. *ACM Transactions on Software Engineering and Methodology, 33*(6), 1-38. [CrossRef]

[31] Granados, Y., Snoeck, M., Ruiz, J., & Ferreira, G. (2024). Experiences from combining Merode and Scrum. In *Agil-ISE@CAiSE* (pp. 27–35).

[32] Tetteh, S. G. (2024). Empirical study of agile software development methodologies: A comparative analysis. *Asian Journal of Research in Computer Science, 17*(5), 30-42. [CrossRef]

[33] Natarajan, T., & Pichai, S. (2024). Behaviour-driven development and metrics framework for enhanced agile practices in scrum teams. *Information and Software Technology, 170*, 107435. [CrossRef]

[34] Ugwueze, V. U., & Chukwunweike, J. N. (2024). Continuous integration and deployment strategies for streamlined DevOps in software engineering and application delivery. *Int J Comput Appl Technol Res, 14*(1), 1-24. [CrossRef]

[35] Suhartini, S., Suef, M., Ciptomulyono, U., & Widodo, E. (2023). A conceptual framework to agile product development for sustainable garment product. In *E3S Web of Conferences* (Vol. 465, p. 02024). EDP Sciences. [CrossRef]

[36] Rubin, K. S. (2012). *Essential Scrum: A practical guide to the most popular Agile process*. Addison-Wesley.

[37] Wiratama, J., & Santoso, H. (2023). Developing a Class Scheduling Mobile Application for Private Campus in Tangerang with the Extreme Programming (XP) Model. *G-Tech: Jurnal Teknologi Terapan, 7*(2), 484-493. [CrossRef]

[38] Melnyk, K. V., Hlushko, V. N., & Borysova, N. V. (2020). Decision support technology for sprint planning. *Radio electronics, computer science, control*, (1), 135-145. [CrossRef]

[39] Seniv, M. M. (2023). Method for Selecting a Software Development Methodology Taking into Account Project Characteristics. *Radio Electronics, Computer Science, Control*, (2), 134-134. [CrossRef]

[40] Fageha, M. K., & Aibinu, A. A. (2013). Managing project scope definition to improve stakeholders' participation and enhance project outcome. *Procedia-Social and Behavioral Sciences, 74*, 154-164. [CrossRef]

[41] Nuti, V. (2023). *Analysis of the effectiveness of the Scrum approach in the management of an IT project* (Doctoral dissertation, Politecnico di Torino).

[42] Merzouk, S., Jabir, B., Marzak, A., & Sael, N. (2024). Best agile method selection approach at workplace. *Bulletin of Electrical Engineering and Informatics, 13*(3), 1868–1876. [CrossRef]

[43] Moyano, C. G., Pufahl, L., Weber, I., & Mendling, J. (2022). Uses of business process modeling in agile software development projects. *Information and Software Technology, 152*, 107028. [CrossRef]

[44] Kalem, G., Vesek, M. C., & Yalim, H. K. (2023, February). The Efficiency of Software Methodologies Used in Artificial Intelligence-Based Biomedical Projects. In *International Congress on Information and Communication Technology* (pp. 615-625). Singapore: Springer Nature Singapore. [CrossRef]

[45] Papadakis, E., & Tsironis, L. (2018). Hybrid methods and practices associated with agile methods, method tailoring and delivery of projects in a non-software context. *Procedia computer science, 138*, 739-746. [CrossRef]

[46] Herda, T., Pichler, V., Zhang, Z., Abrahamsson, P., & Hanssen, G. K. (2025, June). AI and Agile Software Development: From Frustration to Success XP2025 Workshop Summary. In *International Conference on Agile Software Development* (pp. 3-13). Cham: Springer Nature Switzerland. [CrossRef]

[47] Santos, E. P., Gomes, F., Freire, S., Mendonça, M., Mendes, T. S., & Spínola, R. (2022, November). Technical debt on agile projects: Managers' point of view at stack exchange. In *Proceedings of the XXI Brazilian Symposium on Software Quality* (pp. 1-9). [CrossRef]

[48] Alam, M. M., & Priti, S. I. (2024). *Adaptive Hybrid Software Project Management in Bangladesh's Software Industry: Navigating the Cultural Transformation and Ensuring On-Time Delivery* (Doctoral dissertation, IUB).

[49] Bose, B., Khan, N. T., Ashreen, S., Ahmed, F., Mazid-Ul-Haque, M., & Bhowmik, A. (2023). Hybrid scrum-xp: A proposed model based on effectiveness of agile model on varieties of software companies in

bangladesh.

[50] Wandri, R., Fadhilah, M. R., Setiawan, P. R., & Fadhilla, M. (2025). Agile Scrum as a development approach: A case study of web-based school information system design. *Sistemasi: Jurnal Sistem Informasi, 14*(4), 1722–1735. [CrossRef]

[51] Abrar, M. F., Alferaidi, A., Almurayziq, T. S., Saqib, M., Khan, W., Khan, Z., & Alsaffar, M. (2025). Enhancing Extreme Programming (XP) adoption through SAMAM: A scalable agile maturity assessment model based on industry best practices. *IEEE Access*. [CrossRef]

[52] Mustafa, N., Saeed, S., Abdulhakeem, A., & Ibrahim, M. A. M. (2023). The impact of Scrum-XP hybrid methodology on quality in web development with distributed teamwork. In *Proceedings of the 3rd International Conference on Emerging Smart Technologies and Applications* (*eSmarTA*) (pp. 1–8). IEEE. [CrossRef]



Samia Akhtar received her M.S. degree in Computer Science from Virtual University of Pakistan, Lahore. Her research interests lie in the fields of Software Engineering, Machine Learning and Deep Learning. (Email: samiaakhtar9898@gmail.com)



Shabib Aftab completed his Ph.D. in Computer Science from National College of Business Administration and Economics, Lahore. He previously received the M.S. degree in Computer Science from Comsats University, Islamabad, the M.Sc. degree in Information Technology from the Punjab University College of Information Technology, Lahore, and the Bachelor's degree from Government College University, Lahore. He is a dedicated academician and accomplished researcher with over 16 years of academic and research experience. Currently, he is serving as an Assistant Professor in the Department of Computer Science at the Virtual University of Pakistan. His research interests include Software Process Improvement, Data Mining, Predictive Analytics, Applied Machine Learning, and Applied Data Science. Dr. Aftab has authored more than 60 research publications in prestigious international journals and conferences. His work focuses on developing innovative, data-driven solutions to real-world challenges across domains such as healthcare, finance, and software engineering. He is also a Senior Member of IEEE, reflecting his ongoing commitment to advancing research and contributing to the global scientific community. (Email: shabib.aftab@gmail.com)