

Agile Methodologies in Software Development: An Applied Review for Contemporary Engineering Practice

Lawal, K.H

Software Engineering Department, Federal University of Technology Minna, Nigeria

Date of Submission: 12-02-2026

Date of Acceptance: 25-02-2026

ABSTRACT - Agile software development process has gained importance in modern software engineering because of its capability to address rapidly evolving requirement, facilitate customer participation and increase delivery productivity. In stable project environments, plan-driven processes work well; in dynamic and uncertain contexts they tend to lose their effectiveness. This paper is a brief applied review of agile methods; it draws upon peer-reviewed contemporary literature to provide an extended overview for the evolution, principles, practices and connections to engineering and management context. Key agile frameworks, such as Scrum, Extreme Programming (XP), Kanban, Feature-Driven Development (FDD), Crystal and Adaptive Software Development (ASD) are discussed with a focus on their pragmatic adoption, strengths and weaknesses. In addition, the paper provides a comparison between agile and traditional development methods proposes organizational and scaling issues and shows new trends like large-scale agile as well as DevOps incorporation. It has the potential of helping engineering managers and practitioners make evidence-based decisions when selecting suitable Agile practices for their actual industrial environment by synthesizing existing validated academic and industry evidence on this topic.

KEYWORDS: Agile system development, Scrum, Extreme Programming (XP), Kanban, Software engineering management.

I. INTRODUCTION

The growing complexity of software systems, along with rapidly evolving user requirements and technology advances has made the development of software is a difficult engineering task. Contemporary software development projects are performed in non-determinate world of limited time and scope and the dynamic needs of users. In such an environment, rigid plan-driven software development methods e.g., the Waterfall model; have been demonstrated to be highly constraining

focusing in their lack of flexibility to change once implementation has started [1].

Empirically, failure rates for software projects that are developed in a fixed and sequential way are high: late or over budget delivery of out-of-sync solutions with what the end users actually want is widely established [2]. These problems were the catalyst for agile software development approaches, which focus on flexibility, iterative delivery, tight customer engagement and frequent feedback.

Formal recognition of agile methodologies came with the release of the Manifesto for Agile Software Development in 2001 [3], which defined an ethos that favoured individuals and interactions, working software, customer collaboration, and response to change. Since that time, agile has changed from a handful of lightweight processes used by small co-located teams to a set of practices and frameworks applied across the business world by organizations big or small in any industry.

Agile approaches are also widely adopted, but can be misrepresented and implemented vaguely or with a lack of attention to the organizational context. The swift evolution of agile practices (e.g., large scale agile frameworks, integration with DevOps - development (Dev) and IT operations Ops) also calls for regular academic update. Most of the earlier experiments were overly descriptive nature while non-scholarly sources were thoroughly utilized making them less relevant and unsuitable to current researches and applications.

This paper fills these gaps by offering a comprehensive, informed view of agile methods in an applied context based solely on peer-reviewed and other reputable sources. The study aims to:

- i. examine the evolution of software development methodologies leading to agile approaches;
- ii. analyze major agile frameworks and their practical characteristics;
- iii. compare agile and traditional development methodologies; and

- iv. discuss adoption challenges, scaling issues, and emerging trends relevant to

II. BACKGROUND AND RELATED STUDIES

2.1 Evolution of Software Development Methodologies

It was also while largely unstructured and clouded in myth that early software construction took place, and developers until the late 1960s, early 1970s realised a “software crisis” as they feared that software complexity would overtake their ability to manage it. Projects frequently went over-budget, did not match user needs, or were simply dropped. To rationalise this, structured approaches to software development were born in an effort to bring discipline and certainty to the process of creating software [4].

The Waterfall model was one of the first formal methodologies advocating that process be organized in a linear fashion, from bug-filled vision to final release with little going on in between. Appropriate for projects that are presumed to have no volatile requirements, but later research would show them to be ill-suited when requirements change over time [5].

In order to overcome these limitations, iterative and incremental methods such as the Spiral model and object oriented methodologies, e.g., RUP (Rational Unified Process), were created. These methods added feedback loops, control notations and risk management but still placed a significant effort on documentation and upfront planning [6].

By the 1990s, dissatisfaction with heavyweight

engineering and management practice.

methodologies grew, especially of those practitioners applying faster time-to-market-driven projects. Agile methods were proposed as lightweight process to overcome process weight and improve ability to adapt. Systematic literature reviews support that agile development was deliberately created to resolve the rigidity and delayed feedback in traditional methodologies [7].

2.2 Empirical Research on Agile Methodologies

During the last two decades, there has been a significant amount of empirical research studying agile practices and its effect on software development outcomes. It is stated that there has been demonstrative improvement in productivity rates, team communication, and customer satisfaction when adopting agile methods [8].

In spite of its benefits, however, a body of research also identifies significant issues with agile uptake when organizations are large or distributed. Nerur [9] acknowledge that an agile transformation is usually a significant cultural and organizational change, which goes beyond process changes to affect leadership style, governance mechanisms, and reward systems.

More recent research has also turned to large-scale agile adoption, hybrid models and connection to DevOps. These findings indicate that even though agile principles endure, their adoption needs to be tailored to the organizational environment, project size and regulatory requirements [10], [11].

delivers development work in short, time-boxed components called sprints and generally ranging from two to four weeks [12].

Scrum defines three primary roles: the Product Owner, responsible for maximizing product value; the Scrum Master, who facilitates the process and removes impediments; and the Development Team, a cross-functional group responsible for delivering working software. Core Scrum artifacts include the product backlog, sprint backlog, and increment, while events such as sprint planning, daily stand-up meetings, sprint reviews, and retrospectives structure team interaction.

Empirical studies have demonstrated that using Scrum increases transparency, team coordination and stakeholder commitment, especially in organizations with highly volatile requirements [8]. Although, Scrum lacks sufficient guidance for engineering practices and this is likely to result in a disparity of the technical quality without application of disciplined development techniques [7].

III. OVERVIEW OF AGILE METHODOLOGIES

Agile development is not a methodology; it's a collection of philosophies, frameworks and best practices based on the Agile Manifesto. Agile methodologies vary in form and emphasis, but are characterised by their focus on iterative development, customer involvement and feedback, as well as being change oriented. The Most Popular Agile Methods Detailed descriptions of 5 methods with pros, cons and expert recommendations RATE THIS WITH Indeed, many software development methodologies exist which are the same in concept but vary a little bit in practice.

3.1 Scrum

Most popular in small teams and large companies, Scrum is the most common agile framework. It offers a flexible management framework that

3.2 Extreme Programming (XP)

XP (Extreme Programming) is an agile development process that focuses on engineering quality and disciplined coding techniques. First introduced by Beck [13], the XP aims to increase the quality of software and its ability to adapt to changes with practices like test-driven development, continuous integration, pair programming, refactoring, and incremental deliveries.

XP is well suited for projects that experience a high degree of changing requirements and tight customer participation. Its focus on constant testing and feedback leads to lower defect rates and increased maintainability. Research results indicate that XP practice can significantly improve code quality and team learning, as long as it is consistently used [14]. However, XP is predicated on the availability of very skilled developers and unfailing customers. These assumptions restrict its applicability to large teams or distributed cases where coordination and communication overhead is higher [15].

3.3 Kanban

Kanban is an agile methodology based on lean manufacturing principles, concentrating on visualization of workflow, WIP (work in progress) constraints and accounts for a process flow. Unlike Scrum, the approach is not prescriptive of fixed dates (iterations) and roles in the process, stretching its much more flexible in terms of fitting an organization's pre-existing methodology [16].

Work appears on the Kanban board so teams can see bottlenecks. Display-based approaches, such as Kanban, are well-suited to maintenance, support and continuous delivery settings where work arrives in an unpredictable manner [17].

Although Kanban improves process transparency and flow efficiency, opponents argue that the absence of time-boxing and predefined roles may impede predictability and accountability if not controlled [16], [18].

3.4 Feature-Driven Development (FDD)

Feature Driven Development (FDD) is an agile based methodology that stratifies imaginary learning by delivering real, business-valued features actively and iteratively. Proposed by De Luca and Coad [19], FDD integrates object-oriented modelling with feature-based planning and execution.

FDD is model or feature-driven and continually aims to keep its five main activities in coherence: 1) developing an overall model, 2) building a feature list, 3) planning by feature, 4) designing by feature level and by class (using UML), and finally; then and only then implementing the described class

“by-features. That being said, FDD is well adapted for larger projects where domain modelling or feature ordering justify an upfront investment.

One of the goals of FDD is to increase management controllability and documentation, and research shows that it provides more control and documentation than other agile methods [20]. However, it may be less flexible in high volatility scenarios due to its modeling-based foundations.

3.5 Crystal Methodologies

Crystal methodologies, proposed by Cockburn [21], are a family of agile methods that adapt to the project size and criticality. Instead of being pinned down to a specific process, Crystal is centered on people oriented processes and communication and the lessons which can be learned from them.

Crystal methods prop LinkedIn deliveries, osmotic shouting and wall-to-wall posting. The flexibility in Crystal means teams can modify practices to suit their contexts, and so small-to-medium, non-life-critical teams will find it appropriate.

Although Crystal's flexibility is a merit, its non-directional rules may be problematic for inexperienced teams or organisations wishing to adopt more standardized development approaches [22].

3.6 Adaptive Software Development (ASD)

The Adaptive Software Development (ASD) methodology was proposed as a reaction to the growing complexity and unpredictability of software projects [13]. ASD focuses on the adaptive loops of speculation, collaboration and learning, accepting that change is inevitable - and frequently advantageous!

The ASD promotes experimentation, risk-development and continuous improvement. These features make it especially useful for creative and experimental projects in which requirements cannot be completely foreseen.

Nevertheless, ASD is highly dependent on organizational maturity and trust (empowered teams and light governance). These criteria can be challenging to achieve in traditional organizational environments [24].

IV. COMPARATIVE ANALYSIS OF AGILE AND TRADITIONAL DEVELOPMENT APPROACHES

Traditional plan-driven and agile adaptive are two broad categories of software development methods. While these paradigms are conceptually related, comprehension about their fundamental differences is critical to engineering managers and practitioners

when choosing a methodology suitable for the project environment.

Classical software engineering models such as the Waterfall model focus on extensive up-front planning, detailed documentation, and linear execution of development stages. It is based on the premise that systems are completely, specified at a certain stage in time (usually at an early stage), and change little throughout this phase. Although these assumptions may be true in stable and highly controlled environment, we find that they are unrealistic in today's software projects [1].

Agile on the other hand is built to be able to deal with uncertainty and change. Agile is centered on iterative delivery with continuous stakeholder feedback and early deployment. Instead of trying to suppress change, agile processes embrace it as a natural part of software development and use it as means improve on value of the product [3].

4.1 Key Dimensions of Comparison

One of the key differences between agile and nonagile methodologies is how requirements are handled. Traditional development highlights "frozen" requirements and contracts, but agile provides dynamic customer interaction and evolving reqs. Empirical evidence has demonstrated that agile project teams are more resilient to requirement changes and therefore are aimed at a closer alignment with users [25].

Another critical dimension is documentation. In the traditional approaches development is driven by the sheer amount of documentation and for long time, used to be maintained on same documentation line. Agile methodologies, as opposed to eliminating documentation outright, emphasize working software as the most essential sign of progress. Studies show that the move is proven to make students developed faster and with lower cost, but an added risk of knowledge retention how far will them remember if there is less information to refer back to [26],[27].

The structure of the teams and style of management are also poles apart. Classical methodologies frequently use hierarchical, control-based management hierarchies agile methods promoting self-organizing, cross-functional teams. Research shows team process gains (motivation, communication and collective ownership) based on agile teams have a positive influence on performance [8],[28],[29].

4.2 Summary Comparison (shown in Table 1).

In practice Agile methods are found to work better than traditional approaches in highly uncertain change environments with frequent customer

contact. However, for projects where the requirements change less often or have strict to safety-critical regulatory constraints traditional methods might still be applicable [4].

Table 1: Summary Comparison

Dimension	Traditional Methods	Agile Methods
Requirements	Fixed, defined upfront	Evolving, adaptive
Planning	Long-term, predictive	Short-term, iterative
Documentation	Extensive	Lightweight
Customer involvement	Limited	Continuous
Team structure	Hierarchical	Self-organizing
Change management	Change-resistant	Change-embracing
Risk handling	Late-stage	Continuous

V. BENEFITS, CHALLENGES, AND ADOPTION ISSUES

5.1 Benefits of Agile Adoption

There have been several independent studies detailing the benefits of agile Read more > 19 practices in empirical practice. They provide advantages such as faster time - to - market, better product quality and customer satisfaction, and increased team spirit [30]. Agile methods promote quick discovery of bugs due to continuous testing and regular integration, saving a lot of rework & maintenance cost.

When it comes to project management, agile approaches increase the visibility of projects and stakeholders. Regular reviews, demonstrations provide early feedback and support informed decision making to mitigating the risk of developing wrong products [31], [32], [33].

5.2 Challenges and challenges of implementation

However, agile methodologies have several adoption difficulties of their own. Here, organizational resistance to change is one of the largest obstacles. Adopting 'agile' can mean changes in management style, performance measurement and organisational culture—all of which might be opposed by both managers and employees [9], [34].

Other frequently articulated issues are lack of training, lack of agile experience and challenges in amalgamating agile with existing organisational

practices. Two, distributed teams can face communication and coordination challenges which can hinder the effective use of agile practices [30].

5.3 Readiness in the Organization and Management Implications

Successful agile application highly depends on how ready the organization is. Leadership commitment, learning and governance support Research stresses the role of leadership commitment, continuous learning and supportive governance structures in fostering agility increasingly becoming a popular [8],[28],[29]. Engineering managers are instrumental in enabling this transition through encouragement of collaboration, team empowerment, and synchronization of agile practices to meet organizational strategy.

VI. SCALING AGILE AND CONTEMPORARY TRENDS

Agile methodologies emerged in response to the need for nimble development within small, co-located teams, but they have taken root across large and chaotic enterprises as well. When it comes to rolling out agile at scale across teams and departments, new complications of control, compliance and linkage with corporate strategy emerge.

6.1 Large-Scale Agile Frameworks

To mold agility to enterprise environments, there have been proposed large-scale agile frameworks like Large-Scale Scrum (LeSS) and Scaled Agile Framework (SAFe) as means toward mitigating for coordination and governance concerns. These are frameworks to scale the agile principles to more than one team and they describe how teams can be synchronized and have shared backlogs, but also portfolio-level planning [10], [33].

Empirical Work reveals that implementing a framework is not enough for achieving large-scale agile success. Culture, leadership and cross team comments are key for success [30]. Companies who try to scale agile without addressing their cultural and organizational impediments, usually end up less agile, more complicated.

6.2 Agile and DevOps Integration

One popular trend in this direction is the merging of agile techniques and DevOps methodologies. DevOps: CI/CD, Demarcation#Go looking for the standard and acceptance of shared reporting between development and operations within DevOps. When combined with some agile practices, DevOps can improve deployment frequency, software quality and feedback loops [35].

Studies have found that agile–DevOps integration helps close the gap from code to production, while promoting speed and quality of value provided as well as system stability. This integration does however require time and effort in automation, tooling and organizational change which can be a hurdle for organizations lacking resources [36].

VII. LIMITATIONS AND RESEARCH GAPS

Despite this broad success of agile approaches, there are also some limitations and research gaps to fill. First, a majority of the empirical work that has been conducted so far tends to concentrate on small to medium teams in software-rich organizations, raising questions about its generalizability across domains (e.g., safety-critical and regulated industries).

Second, the successes of agile methods are often built on assumptions about team competence, client presence and organizational slack. When these assumptions do not hold, agile practices can lead to less-than-desirable results [26], [27]. Lightweight documentation implies a difficult long-term of the maintenance and the knowledge transfer in case of complex system.

Hybrid development models that combine agile flexibility with traditional discipline models, to be subject of future research. In addition, we need longitudinal studies on the long-term effects of agile adoption regarding impacts on organizational performance and software sustainability.

VIII. CONCLUSION AND FUTURE RESEARCH DIRECTIONS

Agile software development approaches have revolutionized the field of modern software engineering through facilitating flexibility, interaction and incremental results delivery. The current research has provided an extended applied survey of agile methods, reviewing their historical development, major frameworks, relative strengths and organizational impact within the engineering and management realm.

The investigation proves that the agile processes are very useful in conditions of uncertainty and speed changes. But this transition would succeed only if organizations were ready, had a team of talented practitioners and the necessary leadership support. Agile is not a silver bullet for everyone and everything; instead its practices need to be adjusted based on project, culture of the company and regulatory requirements.

Subsequent research efforts should focus on the use of agile beyond traditional software development,

such as in cyber-physical systems, AI projects, or public sector megaprojects. Moreover, additional empirical research is required to investigate the long-term viability of agile practices in complicated business settings.

REFERENCES

- [1]. Pressman, R. S., and Maxim, B. R. (2020). *Software engineering: A practitioner's approach* (9th ed.). McGraw-Hill.
- [2]. Hamid, M., Ahmad, A., and Aimeur, E. (2019). Factors contributing in failures of software projects. *International journal of computer science and network security*, 19(5), 62-77.
- [3]. Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R. C., Mellor, S., Schwaber, K., Sutherland, J., and Thomas, D. (2001). *Manifesto for agile software development*. Agile Alliance.
- [4]. Boehm, B. (2006, May). A view of 20th and 21st century software engineering. In *Proceedings of the 28th international conference on Software engineering* (pp. 12-29).
- [5]. Boehm, B. (1988). A spiral model of software development and enhancement. *Computer*, 21(5), 61-72.
- [6]. Kruchten, P. (2004). *The rational unified process: An introduction* (3rd ed.). Addison-Wesley.
- [8]. Moe, N. B., Dingsøyr, T., and Dybå, T. (2010). A teamwork model for understanding agile teams. *Information and Software Technology*, 52(5), 480-491.
- [9]. Nerur, S., Mahapatra, R., and Mangalaraj, G. (2005). Challenges of migrating to agile methodologies. *Communications of the ACM*, 48(5), 72-78.
- [10]. Larman, C., and Vodde, B. (2009). *Scaling lean and agile development*. Addison-Wesley.
- [12]. Schwaber, K., and Sutherland, J. (2020). *The Scrum guide*. Scrum.org.
- [14]. Karlström, D., and Runeson, P. (2005). Combining agile methods with stage-gate project management. *IEEE Software*, 22(3), 43-49.
- [16]. Anderson, D. J. (2010). *Kanban: Successful evolutionary change for your technology business*. Blue Hole Press.
- [17]. Ahmad, M. O., Al-Baik, O., Hussein, A., and Abu-Alhaja, M. (2025). Unraveling the organisational debt phenomenon in software companies. *Computer Science and Information Systems*, 22(1), 369-399.
- [18]. Stray, V., Moe, N. B., and Sjøberg, D. I. K. (2020). Daily stand-up meetings: Start breaking the rules. *IEEE Software*, 37(1), 70-77.
- [19]. Chowdhury, A. F., and Huda, M. N. (2011, December). Comparison between adaptive software development and feature driven development. In *Proceedings of 2011 International Conference on Computer Science and Network Technology* (Vol. 1, pp. 363-367). IEEE.
- [20]. Abrahamsson, P., Salo, O., Ronkainen, J., and Warsta, J. (2002). *Agile software development methods: Review and analysis*. VTT Technical Research Centre of Finland.
- [21]. Ibrahim, Muhammad, Shabib Aftab, Birra Bakhtawar, Munir Ahmad, Ahmed Iqbal, Nauman Aziz, Muhammad Sheraz Javeid, and Baha Najim Salman Ihnaini. "Exploring the agile family: A survey." *IJCSNS* 20, no. 10 (2020): 163.
- [22]. Hansen, Z. N. L. (2011). *On outsourcing and offshoring: Challenges facing management and engineering*. DTU Management.
- [23]. Highsmith, J. A. (2002). *Agile software development ecosystems* (Vol. 13). Addison-Wesley Professional.
- [24]. Highsmith, J. (2000). *Adaptive software development: A collaborative approach to managing complex systems*. Dorset House.
- [25]. Vijayarathy, L. R., and Butler, C. W. (2016). Choice of software development methodologies. *IEEE Software*, 33(5), 86-94.
- [26]. Turk, D., France, R., and Rumpe, B. (2002). Limitations of agile software processes. In *Proceedings of the Third International Conference on Extreme Programming and Flexible Processes in Software Engineering*.
- [27]. Shahin, M., Babar, M. A., and Zhu, L. (2021). Continuous integration, delivery and deployment: A systematic review. *IEEE Access*, 9, 106131-106150.
- [28]. Dybå, T., and Dingsøyr, T. (2020). Empirical studies of agile software development: A systematic review update. *Information and Software Technology*, 123, 106309.
- [29]. Tripp, J. F., Saltz, J. S., and Turk, D. (2021). Thoughts on current and future research on agile and lean software development. *Journal of Systems and Software*, 171, 110826.
- [30]. Dikert, K., Paasivaara, M., and Lassenius, C. (2016). Challenges and success factors for large-scale agile transformations. *Journal of Systems and Software*, 119, 87-108.
- [31]. Dingsøyr, T., Nerur, S., Balijepally, V., and Moe, N. B. (2012). A decade of agile methodologies. *Journal of Systems and Software*, 85(6), 1213-1221.
- [32]. Kuhrmann, M., et al. (2022). Hybrid software and system development in practice. *Empirical Software Engineering*, 27, 1-38.

[33]. Tell, P., Babar, M. A., and Küpper, S. (2021). Scaling agile in large organizations. *Journal of Systems and Software*, 182, 111069.

[34]. Paasivaara, M., Behm, B., Lassenius, C., and Hallikainen, M. (2020). Large-scale agile transformation at Ericsson. *Empirical Software Engineering*, 25, 368–408.

[35]. Forsgren, N., Humble, J., and Kim, G. (2018). *Accelerate: The science of lean software and DevOps*. IT Revolution.

[36]. Wiedemann, A., Forsgren, N., Wiese, M., and Krömer, H. (2019). DevOps and organizational performance. *IEEE Software*, 36(6), 28–36.