Asian Journal of Research in Computer Science

Volume 17, Issue 5, Page 30-42, 2024; Article no.AJRCOS.113678
ISSN: 2581-8260

Empirical Study of Agile Software Development Methodologies: A Comparative Analysis

Samuel Gbli Tetteh [a*]

[a] *D Jarvis College of Computing and Digital Media, DePaul University, Chicago, USA.*

Author's contribution

This work was carried out by the author. The author has read and approved the final manuscript.

Article Information

DOI: 10.9734/AJRCOS/2024/v17i5436

Open Peer Review History:
This journal follows the Advanced Open Peer Review policy. Identity of the Reviewers, Editor(s) and additional Reviewers, peer review comments, different versions of the manuscript, comments of the editors, etc are available here: https://www.sdiarticle5.com/review-history/113678

Review article

Received: 19/12/2023
Accepted: 24/02/2024
Published: 27/02/2024

ABSTRACT

The comparative analysis of software development models, also called the Software Development Life Cycle (SDLC), is an everyday discourse among software engineers, reflecting the dynamic nature of the field. Within this realm, various software development methodologies, such as prototyping, spiral development, and Rapid Action Development, have been established and recognised for their unique approaches to software creation. In recent years, Agile methodologies have emerged as prominent contenders in software development, offering flexibility, adaptability, and efficiency in delivering high-quality software within designated timeframes. Among the array of Agile methodologies, including Dynamic System Development Method (DSDM), Scrum, Feature-Driven Development (FDD), Extreme Programming (XP), Kanban, Adaptive Software Development (ASD), Mendix, Lean, and Crystal, several have garnered significant attention in the software development community. Specifically, ASD, DSDM, XP, FDD, Kanban, and Scrum have emerged as prominent choices among Agile methods utilised by software developers. This study conducts a comprehensive examination and comparison of these six Agile software models, aiming to elucidate their functionalities, strengths, and weaknesses. The findings of this comparative analysis seek to provide valuable insights for software industries, enabling informed decision-making when selecting

**Corresponding author: E-mail: samuel.gtetteh@gmail.com;*

software development models for upcoming projects. By understanding each Agile methodology's nuanced differences and capabilities, software developers and industry stakeholders can align their project requirements with the most suitable software development approach, ultimately optimising project outcomes and software quality.

Keywords: Extreme programming; dynamic system development method; feature-driven development; adaptive software development; agile methodology and kanban.

1. INTRODUCTION

Recently, Agile Software Development (SD) methodologies have emerged as a cornerstone in the software engineering landscape, fundamentally transforming how software projects are conceptualised, developed, and delivered. The traditional software development models, such as Waterfall, Prototyping Model, and Rapid Development, no longer suffice in today's dynamic business environment characterised by rapid changes in customer requirements [1].

Software development methods and life cycles are pivotal concepts in software engineering, delineating the stages through which software evolves [2]. Agile principles, rooted in customer satisfaction, stakeholder engagement, and collaborative development, have revolutionised the software development paradigm [3]. Methodologies like Extreme Programming (XP), Feature-Driven Development (FDD), Dynamic System Development Method (DSDM), and Adaptive Software Development (ASD) embody the flexible characteristics necessary to accommodate changing requirements, expedite delivery, and enhance software quality.

Agility in software development signifies a system's capacity to adapt to various changes in requirements and environments swiftly. Agile SD processes, characterised by iterative development cycles, are embraced for their unparalleled flexibility [4]. This study aims to comprehensively analyse Agile SD models, delving into their distinct features, characteristics, and commonalities.

In Agile SD, software requirements, development, and products evolve iteratively to align with dynamic business and customer needs, as shown in Fig. 1. The flexibility inherent in Agile methodologies enables teams to adapt swiftly to changing system requirements, ensuring compatibility with evolving project demands [5]. Agile processes prioritise agility, emphasising the rapid and efficient response to changes across various project dimensions, including needs, budget, schedule, resources, technology, and team dynamics.

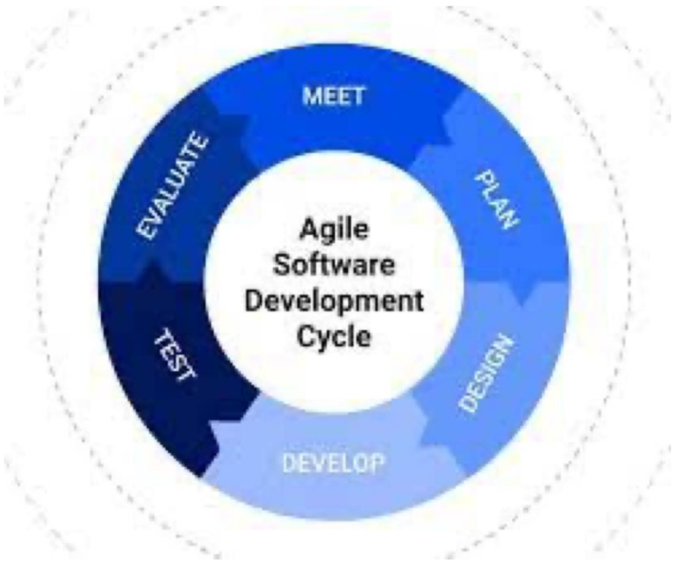

Fig. 1. Agile Software Development Cycle

Central to Agile methodology is the notion that system code is the medium of interaction and documentation between users and computers. The iterative nature of Agile development ensures that functioning software is delivered to customers within agreed-upon timeframes, with new requirements seamlessly integrated into subsequent iterations. By prioritising customer satisfaction, collaboration, and adaptability, Agile SD methodologies have ushered in a new era of software development characterised by responsiveness, efficiency, and continuous improvement

2. LITERATURE REVIEW

Software development encompasses a multifaceted process involving analysis, design, implementation, testing, maintenance, and documentation to deliver software products. Within the realm of software engineering, Agile methodologies have gained significant traction in recent years, becoming integral to industry practices, research endeavours, and scholarly publications. Originating from the concept of iterative enhancement introduced in 1975, Agile methodologies represent a paradigm shift from the rigid and cumbersome nature of traditional developmental processes [6].

Characterised by their evolutionary nature, Agile methods prioritise iterative refinement and adaptive development processes [7]. These methodologies exhibit distinct features, including direct collaboration, adaptability, and incremental deployment. Collaboration between developers and customers is a hallmark of Agile development, facilitating close alignment with customer needs throughout the software lifecycle. Incremental deployment involves the rapid iteration of small software increments, enabling swift responses to evolving requirements. The inherent flexibility of Agile methodologies allows seamless adaptation to changing customer demands during software development.

In contrast to traditional software methodologies, which adhere to linear design, build, and maintenance patterns, Agile development methods emphasise agility and responsiveness to customer requirements [2]. Agile practices, known for their lightweight approach, have emerged as the preferred choice across various industries, enabling effective management of iterative requirements [8]. In today's dynamic technological and business landscapes, traditional software development methodologies often struggle to meet the evolving demands of advanced fields [9].

The Agile software methodology underscores the importance of developer-customer interaction throughout the software development lifecycle. Widely embraced since its formal inception in 2001, Agile methodologies prioritise customer involvement to ensure software products align with customer needs and expectations [10]. Agile methodology encompasses a diverse family of lightweight software development approaches, including Adaptive Software Development (ASD), Lean Software Development, Feature Driven Development (FDD), Dynamic Systems Development Method (DSDM), Crystal, and Scrum [11].

In Agile practices, emphasis is placed on interpersonal interaction over rigid processes and tools. The delivery of working software takes precedence over exhaustive documentation, while collaboration with clients to understand requirements supersedes contract negotiation. Moreover, Agile methodologies prioritise responsiveness to changes, enabling teams to adapt swiftly to evolving project dynamics. This contrasts with traditional software development methodologies' linear and inflexible nature, highlighting the transformative impact of Agile principles on software engineering practices.

3. OVERVIEW OF THE MODEL

Different Agile Methods have been introduced over a period of time, and these types work within a particular category of software domain:(a) Scrum (b) XP (c) DSDM (d) FDD (e) ASD (f) Kanban

4. SCRUM

In the landscape of software development methodologies, the Scrum model, introduced by Ken Schwaber in 1995, has emerged as a cornerstone within the broader Agile methodology framework. Distinguished by its collaborative and iterative approach, Scrum encapsulates a set of principles and practices that redefine how teams conceptualise, plan, and deliver software solutions [12].

The inception of Scrum in 1995 marked a pivotal moment in the evolution of agile software development concepts. Ken Schwaber's model has since gained widespread adoption, positioning itself as a leading methodology in the dynamic and ever-evolving realm of software engineering. Scrum is an integral component of the Agile methodology, embodying its core principles. At its essence, Scrum establishes a collaborative environment where team members work collectively towards the timely and cost-effective delivery of software products. Central to Scrum is the concept of a cohesive team wherein members collaborate seamlessly. This collaborative ethos extends beyond developers to include cross-functional roles, fostering an environment where varied skills converge to achieve a shared goal [13]. The methodology has demonstrated exceptional performance in teams ranging from 5 to 7 individuals, a versatility that extends even to individual developers.

Scrum unfolds as an iterative software development model, operating within distinct roles and responsibilities as shown in Fig. 2. The model's heartbeat is the "sprint," a fixed timeframe typically two weeks, during which the team consistently delivers software products. This iterative rhythm contributes to the reliability and regularity of software releases. Primarily positioned as an ideal choice for short-term projects, the Scrum model excels in scenarios where quick adaptability and responsiveness are paramount. Embracing organization-accepted practices and garnering approval, Scrum mitigates the risk of failure, fostering transparent, reliable, and trusting relations between the development team and customers. One of Scrum's distinctive features lies in its client-centric approach. Clients' involvement as the "product owner" ensures they prioritise and address their most crucial requirements. Moreover, Scrum allows for dynamic modifications to requirements throughout the software development process, aligning the end product more closely with evolving customer needs. Scrum is characterised by its emphasis on flexibility, adaptability, and heightened productivity. The model guides team members in delivering superior software products that exhibit flexibility in accommodating continuous changes in environmental requirements [14].

Scrum avoids explicitly identifying features, opting instead for a list of features that can be dynamically adjusted. Each iteration, or sprint, typically lasts between one week and one month, with three to eight sprints preceding the final product release. This iterative release strategy contributes to a continuous refinement of the end product. The Scrum software development method excels in its focus on client satisfaction through iterative cycles known as sprints. Each sprint integrates all software development life cycle phases, including designing, implementation, testing, and customer review [15,16].

In conclusion, the Scrum methodology is a dynamic and adaptable framework that has significantly influenced the software development landscape. Its collaborative ethos, iterative approach, and client-centric focus position it as a powerful tool for teams navigating the complexities of modern software engineering.

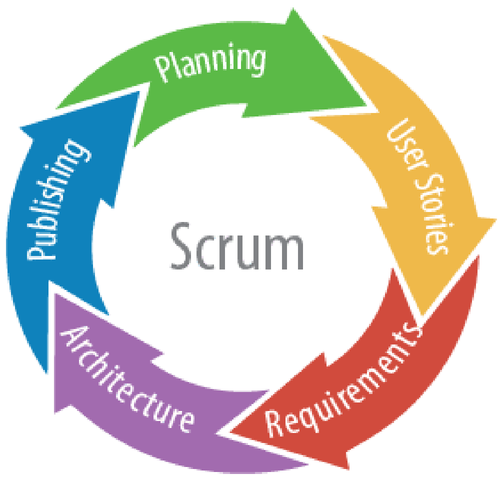

Fig. 2. The Scrum Software Development Cycle

5. EXTREME PROGRAMMING (XP)

Extreme Programming (XP) is a prominent member of agile software methodologies aimed at revolutionising software development practices to enhance quality and responsiveness to evolving customer requirements, as in Fig. 3 below. Rooted in a set of foundational principles, XP embodies a paradigm shift from traditional software development models, emphasising customer-centricity, transparency, and adaptability throughout the development lifecycle. XP emerged as a response to the shortcomings of conventional software development methodologies. It acknowledges the inherent uncertainty in comprehending all functionalities and qualities of complex software systems upfront, advocating for continuous refinement and adaptation as projects progress [17].

At its core, XP embraces key principles underpinning its software development approach. These principles include prioritising customer feedback, embracing simplicity, and welcoming change as integral aspects of the development process. By fostering a culture of collaboration and openness, XP enables teams to respond effectively to evolving requirements and challenges. Designed with small teams in mind, XP thrives in environments where two to ten members collaborate on complex projects. By promoting close collaboration and informal communication channels, XP empowers developers to focus on delivering value rather than navigating bureaucratic processes [18].

In XP, the primary objective is to ensure the success of software development initiatives. This overarching goal permeates every facet of the methodology, driving teams to prioritise continuous improvement, adaptability, and responsiveness to changing requirements. Extreme Programming embodies several core features that distinguish it within the agile landscape. These include small iterations with rapid feedback loops, active customer involvement throughout the development cycle, persistent communication and organisation, continuous refactoring to enhance code quality, seamless integration and testing processes, collective code ownership, and the practice of pair programming [19].

XP unfolds four phases: Design, Code, Test, and Release. Each phase encapsulates a set of practices and rituals geared toward achieving the overarching goals of the methodology: judicious distribution of effort, cost-effective refactoring, and the delivery of correct, high-quality software products within the constraints of small-scale teams. Extreme Programming represents a holistic approach to software development that prioritises collaboration, adaptability, and relentless pursuit of software development success. Its emphasis on customer satisfaction, iterative refinement, and close-knit team dynamics make it a compelling choice for teams navigating the complexities of modern software engineering [20].

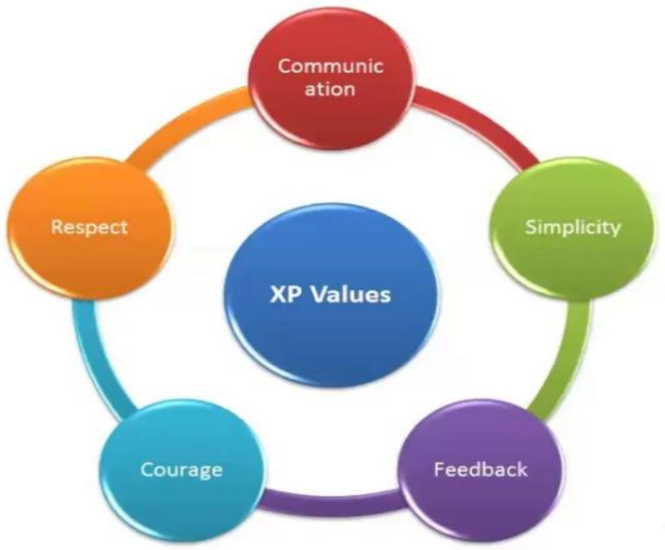

Fig. 3. Extreme Programming Model

6. FEATURE DRIVEN DEVELOPMENT (FDD)

Introduced in 1997, Feature-Driven Development (FDD) emerged as a pivotal component of lightweight, iterative software development methodologies, as shown in Fig. 4 below. FDD represents an iterative and incremental approach to software development, amalgamating the best practices from manufacturing into a cohesive method tailored for software engineering projects. Central to FDD are five fundamental features: advance model, construct feature list, plan feature, design by feature, and build by feature.

FDD epitomises a process-oriented and client-centric agile methodology, placing paramount importance on aligning software development activities with client needs and expectations. By adhering to a structured process flow, FDD aims to deliver software solutions that resonate closely with client requirements while focusing on quality and efficiency. One of the defining characteristics of FDD is its adaptive and incremental nature, which enables the implementation of essential functionality within short, manageable iterations. This iterative approach allows development teams to respond dynamically to evolving project requirements and stakeholder feedback, fostering agility and responsiveness throughout the development lifecycle [3].

At its core, FDD emphasises the significance of design and quality in software development endeavours. By prioritising meticulous design practices and a relentless pursuit of quality, FDD endeavours to deliver frequent and tangible working results at each stage of the system's delivery. This focus on design integrity and quality assurance underpins the methodology's commitment to producing robust and reliable software solutions. FDD provides a structured framework for precise and meaningful advancement throughout software development. By leveraging comprehensive feature lists and well-defined development plans, FDD enables development teams to navigate complex projects with minimal overhead and disruption. This streamlined approach fosters clarity, transparency, and efficiency, empowering designers to make informed decisions and prioritise tasks effectively [21].

FDD distinguishes itself significantly from other methodologies in the development context by its strong emphasis on upfront planning and design. Unlike agile methodologies prioritising flexibility and adaptability over formal planning, FDD advocates for a systematic approach to project initiation and requirements analysis. By laying a robust foundation through meticulous planning and design, FDD sets the stage for successful project execution and delivery [4].

In summary, Feature-Driven Development (FDD) is a comprehensive and structured software development approach characterised by its process-oriented, client-centric philosophy, adaptive nature, emphasis on design and quality, and meticulous planning practices. As software engineering landscapes evolve, FDD remains a compelling choice for teams seeking a systematic and disciplined approach to project delivery.

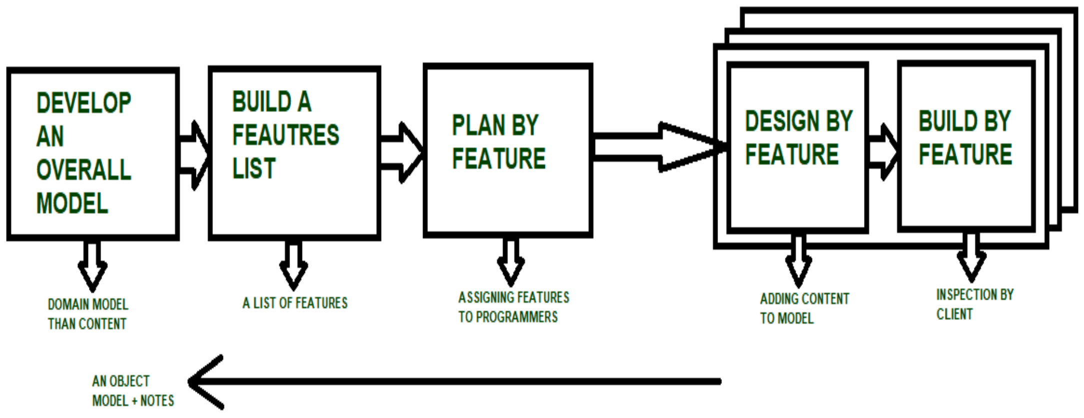

Fig. 4. Feature Driven Development Model

7. DEVELOPMENT SYSTEM DEVELOPMENT METHODOLOGY (DSDM)

Dynamic Systems Development Methodology (DSDM) is a structured software delivery approach widely adopted for developing software packages and non-IT solutions across various domains. It responds to common pitfalls observed in information technology projects, including budget overruns, missed deadlines, and inadequate customer involvement.

DSDM principles revolve around several core tenets aimed at driving successful project outcomes. These principles include a sharp focus on business requirements, timely delivery of software solutions, collaborative teamwork, unwavering commitment to quality, incremental development from solid foundations, iterative progress, continuous communication, and effective project control. The DSDM approach offers a comprehensive framework for developing and maintaining software systems that ensure adherence to project schedules through incremental, iterative prototyping within a well-organized project environment as is shown in Fig. 5. By structuring development efforts around these principles, DSDM seeks to streamline the software delivery process and mitigate project risks effectively [22].

At the heart of DSDM lies a functionality-centric approach to software development, whose primary focus is delivering functional components within predetermined timeframes and resource constraints. Unlike traditional models that fix time and resources and adjust functionality accordingly, DSDM prioritises functionality, adapting time and resources as necessary to achieve project goals. DSDM is widely regarded as a pioneering agile software development methodology rooted in the rapid application development paradigm. It proactively responds to software development teams' collective challenges, particularly project delivery delays and budget overruns [19].

DSDM offers several advantages, including enhanced project transparency, improved stakeholder collaboration, and a structured approach to risk management. Its proactive stance towards addressing common project pitfalls has led to widespread adoption across diverse industries seeking to streamline their software development processes and deliver value-added solutions to end-users [23].

In summary, Dynamic Systems Development Methodology (DSDM) represents a comprehensive and structured approach to software delivery. It is characterised by its adherence to core principles, functionality-centric focus, and proactive stance towards addressing project challenges. As organisations increasingly prioritise agility and efficiency in their software development endeavours, DSDM is a compelling choice for teams seeking to achieve timely, high-quality project outcomes.

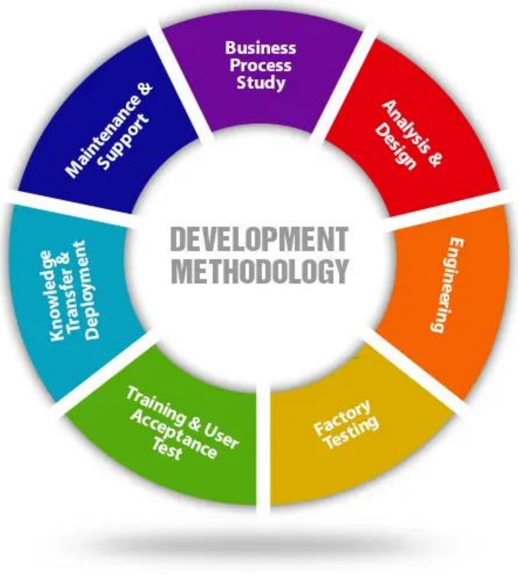

Fig. 5. Development System Development Methodology

8. KANBAN

Kanban, rooted in the Japanese term for visual signs or cards, represents a visual framework utilised to implement Agile principles in software development and other industries. It revolves around the philosophy of Just in Time, aiming to produce the required product of the utmost quality at the precise time and place. At its core, Kanban emphasises workflow visualisation, production duration, and production quantity. It encourages development teams to streamline project workflows, minimise work in progress (WIP) at each stage, and quantify iterations. Unlike rigid methodologies, Kanban inspires incremental modifications within the existing system, fostering adaptability and continuous improvement, as can be seen in Fig. 6 [2].

The versatility of Kanban extends beyond software development, finding applications in manufacturing, logistics, and supply chain management. Its ability to reduce over-production, unnecessary motion, defects, processing, and waiting times has earned it recognition and adoption across diverse systems and industries. A central tenet of Kanban is the management of work-in-progress (WIP) during development. By visualising and mitigating WIP, teams can streamline workflows, prioritise tasks effectively, and enhance productivity [24]. This approach enables precise scheduling and timely delivery of software products to customers, aligning with Agile responsiveness and customer satisfaction principles. Organisations worldwide are increasingly embracing Kanban and integrating it into their existing software development processes to enhance business agility. Kanban mitigates risks, fosters flexibility, and optimises project resource allocation by focusing on the most critical tasks requiring immediate attention [16,25].

In essence, Kanban is a dynamic and adaptable framework that empowers teams to optimise workflows, minimise waste, and deliver customer value efficiently. Its emphasis on visualisation, incremental improvement, and work-in-progress management makes it a valuable tool for organisations seeking to achieve greater agility and responsiveness in today's fast-paced business environment.

9. ADAPTIVE SOFTWARE DEVELOPMENT

Adaptive Software Development (ASD), pioneered by James A. Highsmith, represents a dynamic approach to software development characterised by agility, rapid adaptation, and iterative progress, as shown in Fig. 7. ASD embodies the principles of agile methodology while addressing the challenges posed by high-speed and high-change environments in software projects.

ASD is a tailored iteration of the extreme programming model, one of the most prevalent agile methodologies. Unlike traditional approaches, ASD specifically targets the complexities inherent in large-scale software development endeavours. Its core principles revolve around progressive, stage-wise development supported by stable prototyping techniques. ASD provides a structured framework or methodology that offers sufficient guidance to navigate projects with inherently uncertain requirements. Traditional methods often falter in environments marked by frequent changes in business requirements and rapidly evolving markets. ASD, however, thrives in such dynamic settings by embracing uncertainty and facilitating adaptability [26].

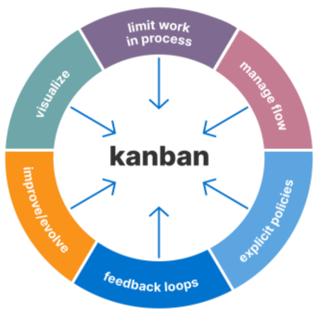

Fig. 6. Kaban Software Development Model

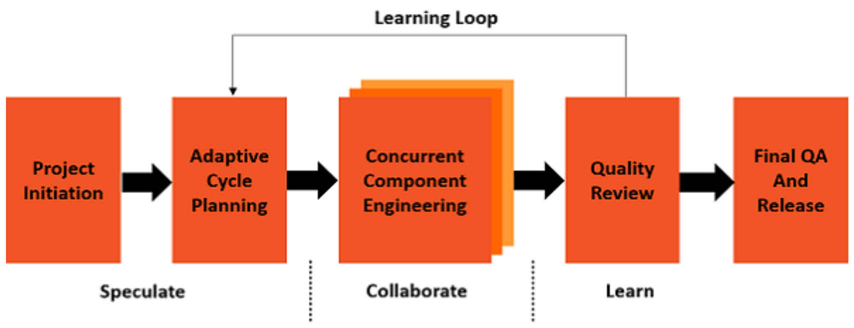

Fig. 7. Adaptive Software Development Model

At the heart of ASD lies continuous learning and extreme teamwork among developers, testers, and customers. Unlike methodologies that prioritise tasks, ASD strongly emphasises products and their quality. It fosters a culture of collaboration, where stakeholders engage in iterative development, testing, and refinement cycles to deliver high-quality software solutions. ASD advocates for incremental and iterative development methodologies complemented by continuous prototyping. This approach allows teams to respond swiftly to changing requirements and market dynamics while ensuring that software products evolve in alignment with stakeholder expectations [27].

In summary, Adaptive Software Development (ASD) represents a paradigm shift in software development methodologies, offering a flexible and adaptive framework for addressing the challenges of today's dynamic business landscape. With its focus on continuous learning, collaboration, and iterative progress, ASD empowers teams to deliver high-quality software solutions in the face of uncertainty and change.

10. COMPARATIVE ANALYSIS OF AGILE METHODOLOGIES

In the realm of software development, choosing a suitable methodology is paramount to project success. With the proliferation of agile methods, it becomes crucial to conduct a comparative analysis to determine which approach best suits a project’s unique requirements. Several factors, including project complexity, size, budget, and time constraints, play a significant role in methodology selection.

11. DOCUMENTATION PRACTICES

Agile methodologies prioritise minimising documentation to focus on delivering working software efficiently. While documentation remains essential, its extent varies across methods. Scrum, XP, and ASD emphasise minimal documentation, prioritising direct communication and working software over extensive paperwork. In contrast, FDD involves more documentation to support its feature-driven approach. DSDM and Kanban strike a balance, requiring moderate documentation compared to FDD.

12. PROJECT COMPLEXITY AND SIZE

Different methodologies are tailored to suit projects of varying complexity and size. Kanban, XP, and ASD are well-suited for simple, small-scale projects with evolving requirements. These methodologies excel in environments where flexibility and adaptability are paramount. Scrum, FDD, and DSDM cater to projects spanning a broad spectrum of complexity, from simple to highly intricate.

13. CUSTOMER INVOLVEMENT AND INTERACTION

Agile methodologies place a strong emphasis on customer involvement throughout the development process. XP and Scrum foster extensive customer interaction, integrating feedback iteratively to ensure alignment with user needs. ASD and DSDM involve clients primarily at the outset and conclusion of iterations, maintaining a continuous feedback loop. Kanban facilitates communication through the product owner, while FDD relies on detailed reports to engage with customers.

14. MEETINGS AND COMMUNICATION

Effective communication among team members is foundational to the success of agile methodologies. Meetings are informal and

Table 1. Comparison of various agile methods

Factors	**Scrum**	**XP**	**FDD**	**DSDM**	**ASD**	**Kanban**
Suitable project size and complexity	**large and complex problems**	**Small and simple project**	**Large scale projects**	**Complex, simple project.**	**Small and simple project**	**Small and simple project**
Documentation	Simple and Basic	Simple and Basic	More than XP, Scrum, Kanban	Highest among all	Moderate	Moderate
Team work	5 to 7 members	2 to 12 members	4 - 20 but fluctuates with complexity	Not specifically address	Not specifically address	Not specifically address
Changes with an Iteration	Not allowed	Allows within their iterations	Allows continually	Allows and reverse	Expected and welcomed	Any time
Transparency	Transparent.	Transparent.	Transparent.	Transparent.	Transparent.	Highly Transparent
Approach	Iterative, Incremental	Iterative, Incremental	Iterative	Iterative	Iterative, Incremental	Iterative, Incremental
Iteration cycle period	2-4 weeks	1-6 weeks	2 days-2weeks	In 20% of total time 80 % of product	4-8 weeks	2 to 4 weeks but focus on continues flow
Concurrent feature development	Possible	Possible	Possible	Possible	Possible	Possible
Major Practices	Scrum meetings	Simplicity, Pair programming Test driven development	Object Modeling, Development by feature, use of UM l diagram	Time boxing, Moscow, Prototyping	Time boxing, Risk Driven, Feature based.	Visualizing processes
User involvement	Through product owner	Actively involved	Through reports	Through frequent releases	Through frequent releases	Through product owner

collaborative, promoting open dialogue and problem-solving. XP encourages pair programming, fostering direct communication and knowledge sharing among developers. FDD and DSDM rely on documentation and reports to facilitate communication, ensuring clarity and alignment. Scrum and ASD prioritise face-to-face interactions, leveraging visualisations and discussions to convey project progress and challenges. Kanban employs visual cues to streamline workflow management, enhancing communication and coordination among team members.In conclusion, a comprehensive understanding of various agile methodologies' distinct characteristics and practices is essential for selecting the most suitable approach for a given project. By considering factors such as documentation requirements, project complexity, customer involvement, and communication practices, project teams can make informed decisions to optimise software development processes and outcomes.

15. CONCLUSION

Agile methodologies have mainly been used for developing software recently compared to traditional software development methodologies. Traditional development practice has countless limitations, which include being unable to adapt to common changes in user requirements and being incapable of working within a specific time frame and budget. Software development methodologies play a significant role in every software project. Change is necessary in software development activity and can occur due to constant changes in user requirements, which makes the agile method the most comprehensive methodology to be adopted. expectedrequirements must be specified clearly before development in traditional software development because it does not adhere to frequent changes. Considering the changing business environment, it is vital that the development methodology used easily adapts to the frequent changes in end-user demands. However, we discussed six agile methods: Scrum, XP, FDD, DSDM, ASD and Kanban. We strongly believe choosing the best method out of the rest for a specific project is paramount.

16. FUTURE DIRECTIONS IN AGILE SOFTWARE DEVELOPMENT

Future directions for the agile software development model are poised to evolve in response to emerging trends and challenges in the software development landscape. Several key areas warrant attention and innovation further to enhance the effectiveness and applicability of agile methodologies:

- Scaling Agile Practices: As organisations increasingly adopt agile methodologies across larger teams and complex projects, there is a growing need to scale agile practices effectively. Future directions may focus on developing frameworks and tools tailored to support scalability while preserving agility and collaboration across distributed teams.
- Integration with DevOps: Integrating agile principles with DevOps practices is gaining momentum, aiming to streamline the software development lifecycle and enhance collaboration between development and operations teams. Future directions may explore deeper integration between agile and DevOps methodologies to facilitate continuous software solution integration, delivery, and deployment.
- Agile for Non-Software Projects: While agile methodologies have traditionally been associated with software development, there is increasing interest in applying agile principles to non-software projects across various industries. Future directions may involve adapting agile practices to domains such as marketing, finance, and project management to improve agility, responsiveness, and customer value delivery.
- Enhanced Data-Driven Decision Making: Agile methodologies emphasise empirical feedback and continuous improvement. Future directions involve leveraging advanced analytics, machine learning, and artificial intelligence techniques to derive actionable insights from project data. This data-driven approach can enable teams to make informed decisions, optimise processes, and enhance project outcomes.
- Focus on Team Dynamics and Well-being: The success of agile teams hinges on effective collaboration, communication, and team dynamics. Future directions may prioritise initiatives to foster psychological safety, diversity, inclusion, and well-being within agile teams. Emphasising team cohesion and

satisfaction can increase productivity, creativity, and project success.

- Adapting to Remote Work Environments: The global shift towards remote work has necessitated adaptations in agile practices to accommodate distributed teams and virtual collaboration. Future directions may explore innovative approaches, tools, and techniques for facilitating effective remote agile ceremonies, communication, and cooperation while maintaining team cohesion and productivity.
- Continuous Evolution of Agile Principles: Agile methodologies are founded on adaptability, responsiveness, and continuous improvement principles. Future directions will likely involve evolving and refining agile principles to address emerging challenges, seize opportunities, and accommodate evolving business and technology landscapes.

In conclusion, the future of agile software development holds promising avenues for innovation, collaboration, and continuous improvement. By embracing emerging trends and challenges, agile methodologies can continue to serve as a cornerstone for delivering high-quality, customer-centric software solutions in an ever-changing world.

COMPETING INTERESTS

The author has declared that no competing interests exist.

REFERENCES

1. Butt SA. Paci fi c Science Review B : Humanities and Social Sciences Study of agile methodology with the cloud, Pacific Sci. Rev. A Nat. Sci. Eng. 2016;2(1):22–28.
2. Saleh SM, Rahman MA, Asgor KA. Comparative Study on the Software Methodologies for Effective Software Development. International Journal of Scientific & Engineering Research. 2017;8(4):April-2017, no. May..
3. Nawaz Z, Aftab S, Anwer F. Simplified FDD Process Model, I.J. Mod. Educ. Comput. Sci. 2017, 9, 53-59 Publ. Online Sept. 2017 MECS
DOI 10.5815/ij, no. September, pp. 53–59, 2017.
4. Shama PS. A Review of Agile Software Development Methodologies, Int. J. Adv. Stud. Comput. Sci. Eng. IJASCSE 2015;4(11). 2015;4(11):1–6.
5. Qureshi MRJ. An adaptive software development process model," no. August 2008; 2018.
6. Ibrahim M, Khan MJ, Salam A. Comparative analysis of scrum and XP in Pakistani software industry. 2017;2(3):199–215.
7. A. Model. A Survey of Agile Development Methodologies. 2007;209–227.
8. Khan A. The Impact of Agile Methodology (DSDM) on Software Project Management The Impact of Agile Methodology (DSDM) on Software Project Management," no; March, 2018.
9. Moniruzzaman ABM, Hossain SA. Comparative Study on Agile Software Development Methodologies," no. May 2014; 2013.
10. Iyawa GE, Coleman A. Customer Interaction in Software Development: A Comparison of Software Methodologies Deployed in Namibian Software Firms. 2016;1–13.
11. Kumar G. Comparative Analysis of Software Engineering Models from Traditional to Modern Methodologies; 2014.
12. Blom M. Is Scrum and XP suitable for CSE Development ? Procedia Comput. Sci. 2012;1(1):1511–1517.
13. Umbreen M, Abbas J, Shaheed SM. A Comparative Approach for SCRUM and FDD in Agile. 2015;2015(2):79–87.
14. Kumari U, Upadhyaya A. Comparative Study of Agile Methods and Their Comparison with Heavyweight Methods in Indian Organizations. 2013;6:June.
15. Aitken A. A Comparative Analysis of Traditional Software Engineering and Agile Software Development; 2013.
16. Matharu GS. Empirical Study of Agile Software Development Methodologies : A Comparative Analysis," no. May 2019; 2015.
17. Fojtik R. Procedia Computer, Procedia Comput. Sci. 2011;3:1464–1468.
18. Maurer F, Martel S. Extreme Programming," no. February; 2002.
19. Abrahamsson P, Warsta J, Siponen MT, Ronkainen J. New Directions on Agile Methods : A Comparative Analysis; 2003.
20. Qureshi MRJ. Comparison Of Agile Process Models To Conclude The

Effectiveness Comparison Of Agile Process Models To Conclude The Effectiveness For Industrial Software Projects," no. December; 2016.

21. Bukhari JA, Shaheed SM. A Comparative Approach for SCRUM and FDD in Agile," no. January; 2016.
22. Sani A, Firdaus A. A Review on Software Development Security Engineering using Dynamic System Method (DSDM). 2013;69(25):37–44.
23. Chapram SB. An appraisal of agile DSDM approach. 2017;4(3):512–515.
24. Wakode RB, Raut LP, Talmale P. Overview on Kanban Methodology and its Implementation. 2015;3(2):2518–2521.
25. Lei H, Ganjeizadeh F, Jayachandran PK, Ozcan P. Robotics and Computer-Integrated Manufacturing Full length Article A statistical analysis of the effects of Scrum and Kanban on software development projects, Robot. Comput. Integr. Manuf. 2017;43: 59–67.
26. Kaushik A. DSDM and ASD Agile Methodologies. 2016;3(9):2393–2394.
27. Merzouk S, Elhadi S, Ennaji H, Marzak A, Sael N. A Comparative Study of Agile Methods : Towards a New Model-based Method. Int. J. Web Appl. Vol. 9 Number 4 December 2017;9(4):121–128.