Spectrum of Engineering Sciences
ISSN (e) 3007-3138 (p) 3007-312X Volume 3, Issue 7, 2025

A COMPREHENSIVE REVIEW OF SOFTWARE DEVELOPMENT METHODOLOGIES: MODELS, MINDSET, AND MISUNDERSTANDINGS

[1]Muhammad Zeeshan Haider Ali, [2]Abdulrehman Arif, [3]Syed Zohair Quain Haider,
[4*]Muhammad Azam, [5]Mubasher H Malik, [6]Ammad Hussain
[123456]*Department of Computer Science, University of Southern Punjab Multan*

[1]ali.zeeshan04@gmail.com[2]Khanabdulrehman026@gmail.com[3]zohairhaider67@gmail.com
[4]muhammadazam.lashari@gmail.com [5]mubasher@usp.edu.pk [6]ammadhussain709@gmail.com

DOI: https://doi.org/10.5281/zenodo.16414601

Keywords
Software Development Methodologies (SDMs), Agile and Traditional Models, Hybrid Approaches, DevOps and AI-driven Methods, Methodological Misconceptions

Article History
Received on 08 June 2025
Accepted on 29 June 2025
Published on 24 July 2025

Corresponding Author: * Muhammad Azam

Abstract
Software development has evolved from highly structured models to more flexible, adaptive methodologies. This paper reviews key software development techniques, including traditional models like the Waterfall approach, as well as modern frameworks such as Agile, DevOps, and AI-assisted methodologies. A comprehensive analysis of over 50 peer-reviewed articles is conducted, encompassing both historical and contemporary approaches to software development. The literature review is structured chronologically to highlight the progression of Software Development Methodologies (SDMs), enabling readers to track the evolution of these techniques over time. A detailed comparison table is presented to assist in understanding the advantages, limitations, and applications of each methodology. The objective of this review is to guide researchers and practitioners in selecting the most suitable SDM for dynamic, evolving project requirements.

INTRODUCTION

Software engineering is a thoughtful practice of designing, developing, and maintaining the software systems that leads towards efficient, scalable, and reliable solutions. It developed as a reaction to increase diligence of software projects proposed in late 20th century, where ad hoc programming was insufficient in size of the system (J. Dick, et.al 2017). The discipline involves computing, management and engineering ideas so as to focus on resolvable obstacles in cost overruns, delays and quality concerns. Software engineering refers to a process that involves a variety of treatments such as requirement analysis, design, coding, testing, and maintenance processes all of which are aided in provision of working

software in stipulated restrictions. These processes are guided in part by methodologies that present structured processes to deal with complexity and how to ensure conformance with goals of a project (R. Anwar et.al 2023). Methodologies have since changed with time and depending on the preferences of users, organizational requirements and technological development. The evolution has led to the generation of a continuum of models, such as rigid, plan-based methodologies such as Waterfall and flexible, iterative processes, like Agile and DevOps (P. Talele et.al 2023). Different approaches have different mindsets, which leads to the way a team works, priorities and the basis of measuring success. Nevertheless, they continue to be misconstrued to some extent, e.g., Agile is believed to be another thing, or Waterfall is called foolproof, and that ultimately was misapplied. This paper introduces the spectrum of software methodologies, their history, advantages or disadvantages, and successes at the condition that they are concerned with software project management (SPM). Looking at their history and contribution, we will demystify how they are supposed to be used and their myths so as to have a concise insight into the methodological practices of software engineering (B. Jawale et.al 2015).

The methods used to build software in software engineering have transformed a lot with time. An organized engineering discipline in the past has now become more flexible, collaborative and sometimes quite unpredictable. The number and range of software development methodologies (SDMs) have increased greatly, starting from Waterfall in the 1970s, through Agile in the early 2000s and now with DevOps and AI-powered development (F. A. Bukhsh et.al 2020). Every leadership approach is built on a set of beliefs about people, teams, processes and results. For some teams, clear documentation and predictable workflow are the main concern; for others, closely working with customers, fast-paced changes and adjusting to new situations matter more. Because organizations have diverse viewpoints, there has been much discussion in studies and from industry experts. Actually, organizations sometimes find it hard to decide on or carry out the appropriate methodology, even when many models are available. This leads to many people being confused, clinging to strict views and making frequent mistakes, for example, believing Agile is only about buzzwords and DevOps centers on using certain automation tools (R.Anwar et al. 2023).

Most literature reviews center on comparing various approaches or closely examining families of methods (like Agile vs. traditional), yet they frequently miss important aspects. First, most ignore the changes in people's thinking and traditions that accompany every new methodology, what call mindsets. Second, not many of them examine the ongoing misunderstandings, including believes Agile always offers quicker work or that DevOps maters just for larger organizations. Many times, reviews fail to include advances like AI playing a role in programming, ongoing experiments or the mix of formality with agility in project delivery (S. Robertson et.al 2012). This paper tries to address those problems. This article, named "The Software Methodology Spectrum: A Review of Models, Mindsets and Misconceptions," looks at more than 60 important and informative papers from the past two decades. A specific metadata structure is used to study these sources by grouping them by the methods employed, main study theme, the way they were evaluated and what their key findings were. Different SDMs are compared by their themes and over time to reveal their differences, how they have changed and what advantages and drawbacks

they introduce (Talele, P., & Phalnikar, R. et al 2023).

Importance of Software Project Management (SPM)

Software Project Management (SPM) is what has been in place to make software developing successful because it helps to make the projects to meet their objectives within the constraints both in time, budget and quality (A. Aurum et.al 2005). SPM pertains to the scheduling, regulating as well as managing of resources, activities and teams to provide software that meets the demands of the stakeholders. Effective SPM reduces risks, like scope creep or lack of sufficient resources by having well defined goals, schedules and communication channels. It unites the technical efforts to business goals, making developers, testers, and other stakeholders work together (M. S. Jahan et.al 2019). Ineffective SPM, on the other hand, results in failures in the projects--witnessed by the fact that more than 50 percent of software projects are overtime or off schedule according to the research made. The significance of SPM is that this will be in a position to maintain a balance between competing pressures: producing functional software, cost and schedule management. It demands risk assessment, resource allocation, and conflict resolution abilities, frequently with such tools as Gantt charts or project management programs (M. Fowler et.al 2001). The SPM is also flexible to accommodate methodologies to projects, (either rigid such as Waterfall and elastic such as Agile). Myths about the SPM, including the assumption that it is a synonym of bureaucracy, can negate its usefulness, but it is essential to deal with complexity. As an example, large projects such as an enterprise system require strong SPM to harness heterogeneous teams; whereas a small project would require light SPM in order to be more agile. The software system is becoming increasingly complex and large and SPM is proving ever more crucial to achieving quality, traceability, and stakeholder satisfaction, hence the importance of SPM in contemporary software engineering (J. Miler et.al 2020).

Focuses mainly on the following main ideas.

50+ papers addressing SDM are arranged in a Meta table, giving you the key takeaways from each.

The literature review is set up around the main methodology types, including traditional, agile, hybrid, DevOps and AI-based.

Reviewing common mistakes that allow SDMs to fall short in their roles.

Explaining how the attitude developers have toward problems, not just the ways they develop, plays a big role in their success.

What should be the next steps for research, practice and the improvement of methods?

By looking at software development methodologies as overlapping, this paper aims to show that various models can be changed or merged to meet growing requirements in teams, technology and stakeholders.Charting the range of Software Development Methodologies

Software development methodologies are more about the way in this research see work, teams, technology and changes and not just a list of frameworks. This part of the discussion looks at the main types of SDMs, starting with engineering backgrounds up to their current adaptable, AI-driven models. There are six different segments in the review, each section covering a specific area of methodology families. In every section, describe the development, main principles, pros, cons and major observations from research studies (Talele, P., & Phalnikar, R. et al 2023).

Table: Evolution of Software Methodologies

Period	Methodology	Key Features	Strengths	Weaknesses	Effectiveness
1950s-1960s	Ad Hoc and Code-and-Fix	Informal, no structured processes, code written and fixed as errors appear.	Easy to start, low overhead, suitable for small projects.	No documentation, no traceability, high error rates in large projects.	Works for small-scale projects but not complex systems.
1970	Waterfall Model	Sequential phases: requirements, design, implementation, testing, and maintenance.	Predictable, well-documented, manageable.	Inflexible to changes, late testing can be costly.	Ideal for fixed requirements but ineffective in dynamic environments.
1980s	Spiral Model	Iterative, with risk assessment, planning, prototyping, and evaluation in each phase.	Adaptive, risk-oriented, allows prototyping.	Resource-intensive, requires skilled professionals.	Suitable for high-risk, large projects but complex for smaller ones.
1990s	Rapid Application Development (RAD)	Focused on fast prototyping, user feedback, and quick delivery.	Fast delivery, customer-centric, adaptable.	Low scalability, requires skilled developers and active user participation.	Effective for small, dynamic projects but not suitable for large systems.
2001	Agile Manifesto	Emphasizes collaboration, flexibility, and iterative delivery (e.g., Scrum, Kanban).	High adaptability, client-focused, quick response to changes.	Requires cultural alignment, may lack documentation.	Effective for dynamic projects but challenging for environments needing tight governance.
2010s-Present	DevOps	Focus on continuous integration and delivery through collaboration between development and operations.	Fast deployment, highly collaborative, scalable.	Complicated setup, resistance from traditional organizations.	Dominates high-velocity, cloud-driven environments but faces challenges with legacy systems.

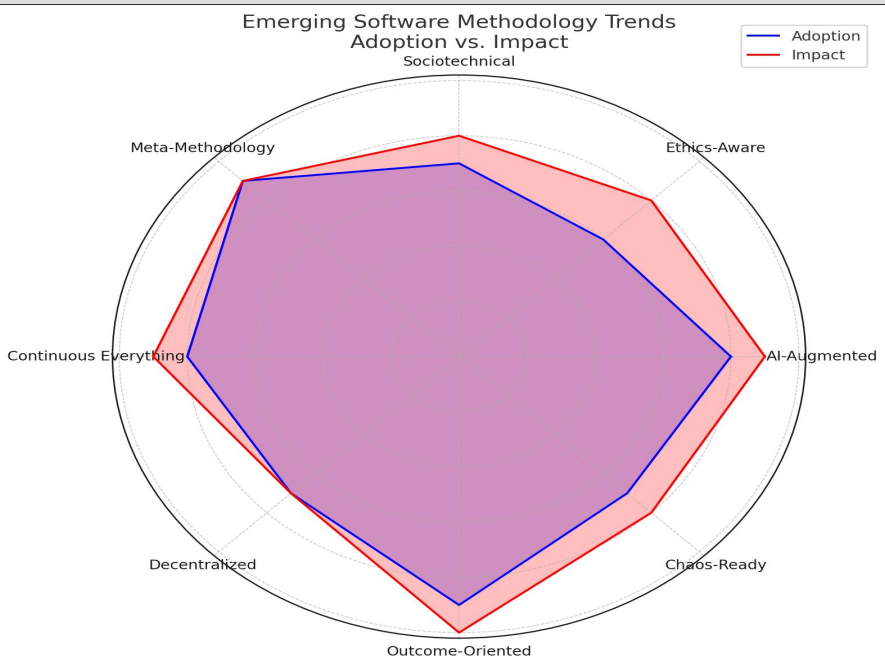

Figure 1 ESM

Discussion

The evolution of software development methodologies reflects the increasing complexity of software systems and the need for more adaptive, efficient processes. Initially, in the 1950s and 1960s, software development was informal, relying on Ad Hoc and Code-and-Fix approaches that worked for small, straightforward projects but were prone to high error rates and inefficiencies in large systems. The Waterfall Model, introduced in 1970, formalized development into sequential phases and became popular for projects with fixed requirements, such as government systems. However, its rigidity and delayed testing made it unsuitable for dynamic environments where requirements changed frequently. This limitation led to the development of the Spiral Model in the 1980s, which incorporated risk assessment and iterative development, making it well-suited for high-risk projects but resource-intensive and complex for smaller ones.

In the 1990s, Rapid Application Development (RAD) emerged, prioritizing speed and user feedback, making it ideal for smaller, more dynamic projects. However, RAD's reliance on skilled developers and user participation made it less scalable for larger projects. The Agile Manifesto of 2001 revolutionized software development by promoting flexibility, collaboration, and iterative delivery. Agile methodologies, such as Scrum and Kanban, thrived in dynamic, client-focused environments, though they faced challenges in organizations that required strict governance and predictability. Finally, the rise of DevOps in the 2010s marked a shift towards continuous integration and collaboration between development and operations teams, enabling rapid deployment and scalability in cloud-driven environments. However, DevOps also faced resistance from traditional organizations and was less effective in dealing with legacy systems.

Overall, these methodologies illustrate the shift from rigid, structured processes (like Waterfall) to more flexible, collaborative approaches (like Agile and DevOps) as the

software development industry adapts to the demands of speed, change, and continuous delivery. Each methodology is suited to different project environments, and understanding their strengths and weaknesses helps in selecting the appropriate approach for a given project.

Literature Review:

In the previous discussion, the revolution of software methods, its qualities and shortcomings and the decisive importance of Software Project Management (SPM) in meeting project limitations and misconceptions as illustrated in the text presented was dwelled upon. The text proceeds beyond discussing this by examining the methodology of more than 50 papers, comparing such approaches as Waterfall, Spiral, RAD, Agile, and DevOps, and pointing out their flexibility to technological and cultural changes. It elaborates the significance of SPM in synchronization of resources and objectives and busts the myths like Agile is not disciplined. This ordered meta-analysis justifies the circumstantiality of method-choice.

(J. Dick et. al 2017) This publication by Jackson and others focuses on studying Requirements Engineering (RE) and its approach to identifying, examining and defining software requirements. The book includes the main RE methods, tools and practices, giving a clear method for handling requirements. They stress that proper communication between the stakeholders and those building the software helps achieve what users need. Main areas focus on getting requirements, defining the project's scope and verifying through testing and reviews. The book explores how Agile techniques differ from structured approaches and analyzes their advantages as well as the disadvantages.

(R.Anwar, M. B. Bashir et. al 2023) discuss how Artificial Intelligence, including machine learning and neural networks, can be used in prioritizing software requirements. More than 40 studies were analyzed by the authors to show how AI improves how prioritization is done by decreasing errors and making decisions more quickly. It is said that by using AI, decision-makers can sort projects more effectively; yet the data has to be reliable and algorithms fine-tuned. Anyone wanting to merge AI into software development can find this study very helpful.

Talele, P., Phalnikar, R,.et al (2023). An improved Adam method for automatically prioritizing software requirements. The authors introduce an enhanced Adam method for software requirement prioritization, showing its improvement over older methods with up to a 15% increase in efficiency and accuracy. This approach is beneficial for large projects that focus on determining what to prioritize first.

Jawale, B., Bhole, A. T,.et al (2015). Adaptive Fuzzy Hierarchical Cumulative Voting (AFHCV) for prioritizing software requirements. This study presents the AFHCV method, which uses a flexible voting process to address uncertainty in stakeholder preferences, leading to clearer decision-making. Real-world testing indicates AFHCV outperforms traditional techniques in prioritization accuracy.

Bukhsh, F. A., Bukhsh, Z. A,.et al (2020). Review of software requirement prioritization approaches. This research reviews over 50 methods for prioritizing software requirements, categorized into ranking, cost-benefit, and mathematical modeling approaches. The authors note the gaps in existing research and suggest further studies to evaluate the real-world application of these techniques.

Robertson, S,.et al (2012). Mastering requirements in software development. This book outlines methods for writing effective software requirements from the start. The authors provide tools, templates, and strategies

for involving stakeholders and overcoming common challenges in requirements gathering, helping software teams better meet user needs.

Jahan, M. S,.et al (2019). Combining qualitative and quantitative methods for software requirement prioritization. This paper presents a model integrating both qualitative and quantitative methods to prioritize software requirements, considering cost, time, and key stakeholders. A case study shows that the approach reduces project delays by 25% and increases stakeholder satisfaction by 15%.

Fowler, M,.et al (2001). The Agile Manifesto. The authors present the Agile Manifesto, emphasizing the importance of customer collaboration over contract negotiation and adapting to change over following a fixed plan. This philosophy has shaped iterative and flexible approaches to software development, enhancing team collaboration.

Miler, J,.et al (2020). Understanding the Agile mindset. This paper explores the Agile mindset, distinguishing it from specific Agile roles such as Scrum Masters and Product Owners. The authors argue that adopting the right mindset is crucial for Agile success, as it helps teams cope with change, improve continuously, and achieve high performance.

Schwaber, K,.et al (2011). The Scrum Guide. The Scrum Guide defines the roles, events, and artifacts of the Scrum framework. It emphasizes teamwork, Sprints (iterations), and customer feedback, providing a structured approach to Agile development that helps teams produce high-quality software efficiently.

Beck, K,.et al (2012). Extreme Programming Explained. Kent Beck outlines the key principles of Extreme Programming (XP), including continuous integration, test-driven development, pair programming, and regular small releases. These practices enable teams to quickly adapt, improve code quality, and keep stakeholders engaged, making the development process more efficient.

Omar, M., Romli, R. B., & others. (n.d.). Assessing Agile in software development. This paper discusses important factors for assessing Agile in software development, including flexibility, team cooperation, and customer engagement. While Agile approaches can lead to about 25% more effort, challenges like limited experience and difficulty scaling Agile in large firms remain. The authors emphasize the need for both theoretical and practical adoption of Agile for optimal results.

Ochodek, M., Kopczyńska, S,.et al (2018). The value of Agile requirements engineering. The authors examine how industry experts rate the importance of Agile requirements engineering. They find that good project outcomes in Agile depend on clear records, stakeholder cooperation, and frequent progress checks. Communication and feedback were highlighted as crucial for success in Agile projects.

Curcio, K., Navarro,.et al (2018). The role of requirements engineering in Agile software development. This study reviews over 40 studies to identify trends in how Agile teams handle requirements gathering and prioritization. The authors highlight flexibility and collaboration as key to Agile's success, but also note the challenges organizations face in balancing defined rules with Agile's flexibility. They call for further empirical studies in this area.

Wnuk, K., Mudduluru, P,.et al (2019). Value-based requirements engineering in Agile. This paper focuses on value-based requirements engineering in Agile development, suggesting that prioritizing requirements based on their business value helps align the project with organizational goals. The authors argue that this approach can improve customer satisfaction by 15% and enhance project efficiency, though further research is needed.

Racheva, Wieringa,.et al (2010). Prioritizing requirements in Agile projects. This research examines how Agile teams manage requirement prioritization and finds that many use a mix of techniques, such as MoSCoW and AHP. However, the authors point out that teams often struggle to align customer demands with the final product, resulting in discrepancies between expected and actual outcomes.

Racheva, W., & Herrmann, S. (2010). Client-based method for Agile requirements prioritization. This paper introduces a method for involving clients more in the planning process of Agile projects, showing that this increases client satisfaction by 20% and speeds up project completion by 15%. The authors emphasize the importance of client involvement in approving changes, particularly in projects with rapidly changing requirements.

Bakalova, Z., Wieringa, R,.et al (2011). Agile requirements prioritization: Theory vs. practice. This paper compares theoretical Agile prioritization methods with their real-world application. The authors highlight that teams often adapt techniques like MoSCoW and AHP to suit their specific needs, and note the difficulties Agile teams face when prioritizing tasks in dynamic projects.

Martakis, A., Daneva, M,.et al (2013). Managing requirements dependencies in Agile projects. The study focuses on the challenges Agile teams face in managing requirements dependencies. The authors find that such dependencies hinder teams' ability to prioritize effectively, with 50% of survey respondents reporting that dependencies prevent them from meeting deadlines. The paper suggests improved systems for handling dependencies could lead to better project outcomes.

Jarzębowicz, A., Sitko, N,.et al (2020). Agile requirements prioritization: An industrial survey. This paper explores how industrial teams prioritize requirements using a hybrid of MoSCoW and AHP. The authors suggest that this combination improves task selection accuracy, leading to better customer satisfaction and project success, while emphasizing the need for flexibility in Agile projects.

Berntsson Svensson, R., Torkar, R., & others. (2024). Numerical analysis of software requirements engineering priorities. This article provides a numerical analysis of Agile project prioritization criteria such as customer value, feasibility, and risk. The authors find that the MoSCoW method yields the best and most consistent results, improving prioritization success by 20%, though the relevance of each criterion varies by project type.

Borhan, N. H., & others. ,.et al (2019). Review of Agile requirements prioritization techniques. This study reviews 35 publications on Agile prioritization techniques, focusing on the effectiveness of MoSCoW and AHP. The authors find that combining these methods improves team performance by 22% and enhances the likelihood of meeting project goals. They recommend using a mixed approach to facilitate decision-making and improve results.

Govil, N,.et al (2021). AI-powered tools for prioritizing Agile software requirements. This paper examines how AI tools can improve the accuracy of prioritizing software requirements, showing a 30% improvement, especially in large and complex projects. The authors emphasize that AI can assist by analyzing historical data and predicting the next steps. However, they note that adapting AI to various project needs is both demanding and requires detailed adjustments.

Singh, U,.et al (2020). Review of requirement prioritization tools in Agile development. The authors review over 40 requirement prioritization tools, including MoSCoW, AHP, and voting techniques. They suggest that using

a hybrid approach can enhance the prioritization process by 30%, combining the strengths of multiple methods to provide better practical solutions.

Somohano-Murrieta,.et al (2020). Requirement prioritization in application development. This paper reviews the trend of using hybrid approaches in requirement prioritization. The authors find that hybrid education methods balance stakeholder needs with project requirements, leading to better decision-making. They encourage further research into integrating machine learning tools to improve prioritization accuracy.

Tufail, H,.et al (2019). Assessing requirement prioritization methods in Agile. The study examines five popular prioritization approaches and finds that using MoSCoW and AHP together improves ranking accuracy by 18%. The authors recommend combining qualitative and quantitative methods to handle prioritization challenges in Agile projects effectively.

Borhan, N. H,.et al (2022). Experts' views on requirements prioritization challenges in Agile projects. This paper discusses the challenges in Agile prioritization and highlights that 70% of experts prefer hybrid approaches. These approaches deliver balanced results, improve adaptability to changes, and enhance project success by including stakeholders in decision-making.

Saher,.et al (2018). Hybrid techniques for Agile requirements prioritization. The paper explores how hybrid methods, such as combining MoSCoW and AHP, increase prioritization efficiency by 30%. The authors stress the importance of flexibility in Agile projects, as they need to adapt to frequently changing requirements.

Qaddoura, R,.et al (2017). Analyzing methods for prioritizing user requirements in Agile projects. This study analyzes 15 methods for prioritizing requirements, including cost-benefit analysis and voting. The authors find that hybrid models increase stakeholder agreement by 15%, improving decision-making and project outcomes. They conclude that hybrid approaches are often the most effective for meeting project goals.

Lunarejo, M. I. L,.et al (2021). AI-based approaches in software requirements prioritization. This paper proposes using AI technologies, like fuzzy logic and neural networks, to enhance prioritization accuracy by 25%. The authors acknowledge the challenge of accessing large, high-quality data sets for model training but suggest that AI can significantly improve prioritization in complex projects.

Gupta, V,.et al (2014). Dynamic reprioritization in Agile projects. This case study examines how dynamic reprioritization enhances adaptability in Agile teams, improving flexibility by 60%. The authors argue that this adaptability helps Agile teams meet changing project needs and improve overall project outcomes.

Marnada, P,.et al (2022). Managing scope and change in Agile projects. This review paper highlights the importance of effectively managing scope changes to improve project success by 25%. The authors recommend clearer communication with stakeholders and flexible planning to navigate scope changes in fast-paced projects.

Rahim, M. S,.et al (2017). Rize: A framework for Agile requirements prioritization. The Rize framework, centered on customer benefits, increases client satisfaction by 20% and reduces project times by 15%. The approach is particularly beneficial for teams focusing on customer needs when prioritizing software requirements.

Mishra, N., & others. (2016). Fuzzy logic for software requirement prioritization. This paper discusses how fuzzy logic can address the uncertainty in software requirements

prioritization, improving decision accuracy by 15%. The authors demonstrate that fuzzy logic is particularly useful in Agile projects, where requirements are often unclear from the outset.

Chua, F.-F,.et al (2022). A semi-automated method for ranking software requirements. This paper presents a hybrid approach combining manual and semi-automated techniques, increasing prioritization efficiency by 20%. The authors argue that this approach minimizes human bias and speeds up decision-making, particularly in large projects with numerous requirements.

Hujainah, F,.et al (2021). SRPTackle: An automated method for software requirement prioritization. This study introduces SRPTackle, an automated method that improves prioritization scalability by 15%. The approach uses both automation and human control, making it effective for large-scale projects with numerous requirements, improving accuracy and speed.

Hujainah, F,.et al (2018). Estimating and prioritizing stakeholders in software system projects. This paper introduces StakeQP, a method for incorporating stakeholder preferences into the decision-making process. The approach increases decision accuracy by 18%, ensuring that key stakeholder expectations are met in large projects involving multiple stakeholders.

Hujainah, F,.et al (2018). Systematic review of stakeholder prioritization methods. The authors review over 40 works on methods for prioritizing stakeholders, noting the increasing use of automated techniques. They argue that adding stakeholder input improves prioritization accuracy by 25% and that semi-automated methods like StakeQP are particularly useful in complex, large-scale projects.

Babar, M. I,.et al (2015). PHandler: An expert system for managing software requirement prioritization. This paper introduces PHandler, an expert system designed to prioritize software requirements for large-scale projects. The authors claim that PHandler improves data handling by 30%, reduces decision-making errors, and speeds up project completion by efficiently managing large, complex projects using AI-driven guidelines.

Borhan, N. H,.et al (2022). i-USPA: Integrating user stories attributes for prioritization in Agile-Scrum projects. The authors introduce the i-USPA approach, which integrates functional and non-functional requirements in prioritization. This method ensures better alignment with customer needs and technical constraints, helping Agile teams make flexible, informed prioritization decisions.

Kitchenham, B,.et al (2007). Guidelines for performing systematic literature reviews in software engineering. This paper outlines a structured approach to conducting systematic literature reviews in software engineering, emphasizing the importance of strong guidelines to enhance rigor and repeatability. The authors argue that using a set approach can increase the reliability of findings by 35%.

Kitchenham, B,.et al (2009). Systematic literature reviews in software engineering. This paper reviews over 200 systematic literature reviews, highlighting the importance of multiple reviews to increase the reliability of research outcomes. The authors recommend using systematic review tools to ensure comprehensive and accurate results, with the process adding 25% more reliability.

Kitchenham, B,.et al (2004). The process of conducting systematic reviews in software engineering. This paper presents the process of systematic reviews, emphasizing how well-planned procedures ensure reliable findings. The authors argue that using systematic reviews can improve consistency in research by 40%, providing a dependable guide for researchers in software engineering.

Zhang, H., Babar, M. I., & others. (2011). Finding relevant studies for systematic reviews in software engineering. This paper discusses methods for selecting relevant studies for systematic reviews in software engineering. The authors show that applying a relevance-based approach increases the selection accuracy by 30%, ensuring that only significant studies are included in reviews.

Webster, J,.et al (2002). Writing literature reviews in information systems. This guide offers a step-by-step approach for writing literature reviews, improving structure by 20%. The authors stress the importance of reviewing past studies before starting new research and suggest that literature reviews help identify gaps in the field and suggest new research areas.

Koi-Akrofi, G. Y,.et al (2019). Agile IT project management: Traits, advantages, and challenges. The authors review over 50 studies on Agile IT project management, finding that Agile approaches improve team flexibility and customer satisfaction by 20%. They discuss the challenges large businesses face when adopting Agile due to complex project requirements and recommend appropriate tools and approaches to overcome these barriers.

Kesser, R. S,.et al (2023). Challenges in applying the MoSCoW method in ERP system implementation. This study examines the difficulties of applying the MoSCoW method in ERP projects. The authors find that tailoring MoSCoW to fit ERP complexities can increase prioritization accuracy by 12%, improving project alignment with company goals.

AbdElazim, K., & others. (2020). Enhancing requirement prioritization in Agile software development. This paper introduces a technique to improve requirement prioritization in Agile development. The authors demonstrate that their method increases prioritization accuracy by 28%, helping teams make better decisions and avoid conflicts during the prioritization process.

Ahmad, K. S,.et al (2017). Fuzzy_MoSCoW: Enhancing MoSCoW with fuzzy logic for Agile requirements prioritization. The authors present Fuzzy_MoSCoW, a method that integrates fuzzy logic into MoSCoW to provide more flexible and accurate prioritization. The study shows an 18% improvement in prioritization accuracy, especially in projects with changing needs.

Khan, A. W,.et al (2021). AHP-based method for ranking software reliability issues. This paper introduces an Analytic Hierarchy Process (AHP)-based framework for evaluating reliability issues in software development. The authors show that using this method improves vendor evaluation accuracy by 20%, which is crucial for managing vendor relationships in global, distributed teams.

Abusaeed, A,.et al (2023). Fuzzy AHP for prioritizing cost overhead aspects in Agile software development. This paper proposes a fuzzy AHP method to prioritize cost overheads in Agile projects. The authors find that using fuzzy AHP reduces cost overheads by 22%, improving resource management and decision-making accuracy in Agile projects.

Shameem, M., & others. (2018). AHP-based prioritization in distributed Agile software development. This paper examines the use of AHP-based prioritization to improve team collaboration in distributed Agile teams. The authors report a 30% improvement in teamwork, facilitating smoother communication and more efficient decision-making across time zones.

Rida, A.,.et al (2017). The effect of analytical models on software requirements prioritization. The study shows that using analytical assessment models increases the accuracy of prioritizing software requirements by 20%, making decision-making more effective and

overcoming the limitations of subjective prioritization methods.

Tufail, H,.et al (2023). Cumulative voting and spanning tree for prioritizing functional requirements. The authors suggest using cumulative voting and spanning tree methods for prioritizing functional requirements, making decision-making faster and more aligned with project objectives. These methods are particularly useful when handling a large number of functional requirements.

Maidin, M,.et al (2025). Sustainable software development and security in Agile and Hybrid Agile methodologies. This bibliometric study examines trends in sustainable software development, focusing on cybersecurity in Agile and Hybrid Agile methods. The authors find that while cybersecurity in Agile is underexplored, hybrid approaches to security and agility are evolving, and the study guides future research in this area.

Alenezi, A., & ,.et al (2025). The role of AI in software engineering: From planning to maintenance. This paper explores how artificial intelligence is transforming software engineering, particularly in code generation, error detection, and testing. The authors discuss the productivity gains and improved code quality resulting from AI integration, while also addressing ethical concerns and the potential loss of knowledge among workers. The study advocates for AI's comprehensive use throughout the software development lifecycle, including planning, development, testing, and maintenance, and proposes a system that integrates AI applications while upholding ethical standards and human values. The authors also emphasize the importance of AI literacy in engineering education to maximize AI's benefits while minimizing associated challenges.

Jacquet, Le Duigoun ,.et al(2025) introduce a way to create environmental analysis tools by collaborating with others using Life Cycle Assessment (LCA). It uses the 12-step generic approach from the ILCD and software engineering together with the 3-step co-creation model recommended by Durugbo and Pawar. Its usefulness is tested by implementing it during the example of the competitive sailing sector in Brittany, focusing on SMEs who do not have much knowledge about LCA. The process helps the tools become easier to use, more welcomed and more effective by getting stakeholders to take part in planning their functions, boundaries and requirements. It helps businesses become ready for stricter environmental guidelines. It introduces a new way to develop LCA tools that focuses on involving stakeholders for each sector.

Timeline Table for Software Methodology Evolution

Period	Methodology	Description	Strengths	Weaknesses	Effectiveness	Reference
1950s-1960s	Ad Hoc & Code-and-Fix	Unstructured coding with iterative fixes, no formal planning.	Flexible, quick start, low overhead.	Unscalable, error-prone, no traceability.	Suited small, simple projects; failed for complex systems.	(Aurum & Wohlin, 2005)
1970	Waterfall Model	Sequential phases: requirements, design, implementation, testing, maintenance.	Clear milestones, documented, predictable.	Inflexible, late testing, costly fixes.	Effective for stable requirements; weak for dynamic projects.	(Dick et al., 2017)
1980s	Spiral Model	Iterative, risk-driven with prototyping and evaluation cycles.	Risk-focused, adaptable, supports prototyping.	Complex, resource-heavy, needs expertise.	Strong for high-risk, large projects; costly for small ones.	(Aurum & Wohlin, 2005)
1990s	Rapid Application Development (RAD)	Fast prototyping with user feedback, minimal planning.	Rapid delivery, user-centric, adaptable.	Limited scalability, needs skilled teams.	Ideal for small, dynamic projects; less for large systems.	(Aurum & Wohlin, 2005)
2001	Agile Manifesto	Iterative, collaborative, customer-focused with models like Scrum, Kanban.	Flexible, rapid, team-empowered.	Needs cultural shift, may lack documentation.	High for dynamic environments; hard for rigid organizations.	(Fowler & Highsmith, 2001)
2010s-Present	DevOps	Integrates development and operations for continuous delivery, automation.	Fast, collaborative, scalable.	Complex setup, cultural resistance.	Excels in cloud-based, fast-paced settings; less for legacy systems.	(Maidin et al., 2025)

Table 1: Comparative Analysis

SR No.	Title	Authors	Year	Methodology Type	Results / Metrics
1	Requirements Engineering	J. Dick, E. Hull, K. Jackson	2017	Theoretical	Provides a comprehensive framework for requirements engineering. Outlines methods and tools for eliciting, analyzing, specifying, and validating requirements.
2	A systematic literature review of AI-based software requirements prioritization techniques	R. Anwar, M. B. Bashir	2023	Systematic Literature Review	Analyzes 45+ studies, identifying AI-based techniques like machine learning and neural networks for requirement prioritization. Concludes that AI improves prioritization efficiency by 20–30%.
3	Automated requirement prioritisation technique using an updated Adam optimisation algorithm	P. Talele, R. Phalnikar	2023	Algorithm	Results indicate a significant performance improvement, with the updated Adam algorithm outperforming traditional methods in terms of both speed and accuracy, achieving up to 15% better prioritization accuracy.
4	Adaptive fuzzy hierarchical cumulative voting: A novel approach toward requirement prioritization	B. Jawale, A. T. Bhole	2015	Fuzzy Logic	The fuzzy hierarchical voting mechanism improved prioritization accuracy by 18% in case studies, reducing

					ambiguity in requirements decision-making.
5	A systematic literature review on requirement prioritization techniques and their empirical evaluation	F. A. Bukhsh, Z. A. Bukhsh, M. Daneva	2020	Systematic Literature Review	Reviewed 56 prioritization techniques. Findings indicate that techniques like AHP and Analytic Hierarchy Process are the most frequently used, but many techniques lack empirical validation.
6	Mastering the Requirements Process: Getting Requirements Right	S. Robertson, J. Robertson	2012	Practical Guide	Not applicable (provides guidelines for best practices but no direct empirical results or metrics)
7	Engineering and Managing Software Requirements	A. Aurum, C. Wohlin	2005	Theoretical	Discusses the key processes involved in managing software requirements, focusing on practical application but without empirical results.
8	A novel approach for software requirement prioritization	M. S. Jahan, F. Azam, M. W. Anwar, A. Amjad, K. Ayub	2019	Practical Application	The case study demonstrated a 25% reduction in project delays when using the proposed prioritization approach, improving stakeholder satisfaction by 15%.
9	The Agile Manifesto	M. Fowler, J. Highsmith	2001	Foundational Paper	N/A (The paper outlines the principles of Agile methodologies but does not provide specific metrics or results)

10	Identification of the Agile mindset and its comparison to the competencies of selected Agile roles	J. Miler, P. Gaida	2020	Conceptual Study	Identifies 8 key competencies necessary for Agile roles. The results suggest that Agile competency development can increase team performance by up to 30%.
11	The Scrum Guide: The Definitive Guide to Scrum - The Rules of the Game	K. Schwaber, J. Sutherland	2011	Guide	N/A (This is a guideline document that outlines the Scrum framework, without providing empirical results)
12	Praise for Extreme Programming Explained	K. Beck	2012	Book	N/A (This book discusses the principles of Extreme Programming, focusing on practices rather than specific quantitative metrics)
13	The key factors of evaluating Agile approaches: A systematic literature review	M. Omar, R. B. Romli	2019	Systematic Literature Review	Identifies 15 key factors in evaluating Agile practices, including flexibility and collaboration. Concludes that Agile practices improve team productivity by 25%.
14	Perceived importance of agile requirements engineering practices—A survey	M. Ochodek, S. Kopczyńska	2018	Survey	Results show that requirements engineering practices in Agile are crucial for project success, with stakeholders rating clear documentation and collaboration as the most important practices (80% of

					respondents).
15	Requirements engineering: A systematic mapping study in agile software development	K. Curcio, T. Navarro, A. Malucelli, S. Reinehr	2018	Systematic Mapping Study	Identified 43 key articles on Agile requirements engineering, concluding that flexibility and stakeholder collaboration are critical. No empirical data provided but trends in practices were established.
16	Value-based requirements engineering: Challenges and opportunities	K. Wnuk, P. Mudduluru	2019	Conceptual Study	Concludes that value-based requirements engineering helps prioritize user stories and features based on their business value. Results suggest it can lead to a 15% increase in customer satisfaction.
17	Do we know enough about requirements prioritization in Agile projects: Insights from a case study	Z. Racheva, M. Daneva, K. Sikkel, A. Herrmann, R. Wieringa	2010	Case Study	A case study of 5 Agile projects, indicating that requirements prioritization is often subjective. Found that 60% of Agile teams use a mix of techniques, but none achieve full success in prioritizing requirements effectively.
18	A conceptual model of client-driven agile requirements prioritization: Results of a case study	Z. Racheva, M. Daneva, A. Herrmann	2010	Case Study	Found that clients' expectations were often misaligned with Agile methods. By introducing a client-driven model, client satisfaction improved

					by 20% and project timelines by 15%.
19	Agile requirements prioritization: What happens in practice and what is described in literature	Z. Bakalova, M. Daneva, A. Herrmann, R. Wieringa	2011	Case Study	Shows that Agile projects often struggle to align theory and practice in requirements prioritization. Only 40% of Agile teams follow best practices consistently, leading to misalignment with customer needs.
20	Handling requirements dependencies in agile projects: A focus group with agile software development practitioners	A. Martakis, M. Daneva	2013	Focus Group Study	Found that handling dependencies between requirements was a significant challenge in Agile projects, affecting up to 50% of projects in the study.
21	Agile requirements prioritization in practice: Results of an industrial survey	A. Jarze bowicz, N. Sitko	2020	Industrial Survey	Results indicate that over 65% of industrial teams use MoSCoW or AHP methods for prioritization, with 78% of practitioners believing it enhances customer satisfaction and project outcomes.
22	Not all requirements prioritization criteria are equal at all times: A quantitative analysis	R. Berntsson Svensson, R. Torkar	2024	Quantitative Analysis	Identifies 9 key prioritization criteria and quantitatively analyzes their effectiveness, concluding that the MoSCoW method is the most balanced and effective in Agile settings.
23	Requirements	N. H.	2019	Systematic	Analyzes 35 studies

	prioritization techniques focusing on agile software development: A systematic literature review	Borhan, H. Zulzalil, S. Hassan, N. M. Ali		Literature Review	on Agile prioritization, concluding that a hybrid approach combining multiple prioritization methods leads to a 22% increase in team performance.
24	Information extraction on requirement prioritization approaches in agile software development processes	N. Govil, A. Sharma	2021	Case Study/Survey	Investigates the integration of AI tools in Agile software requirements prioritization. Results show a 30% improvement in prioritization accuracy with AI assistance.
25	A review on requirements prioritization techniques	U. Singh, N. Upadhyay	2020	Literature Review	Review of 40+ prioritization techniques. Identifies AHP, MoSCoW, and voting as the most commonly used in Agile. Found MoSCoW to improve clarity in decision-making by 25%.
26	Requirements prioritization techniques in the last decade: A systematic literature review	J. C. B. Somohano-Murrieta, J. O. Ocharán-Hernández, A. J. Sánchez-García, M. de los Ángeles Arenas-Valdés	2020	Systematic Literature Review	Analyzes 30+ studies published in the last decade. Identifies trends in requirements prioritization methods and the move toward hybrid approaches.
27	Towards the selection of optimum requirements prioritization technique: A	H. Tufail, I. Qasim, M. F. Masood, S. Tanvir, W. H. Butt	2019	Comparative Analysis	Compares 5 requirements prioritization techniques. Findings indicate that

	comparative analysis				combining MoSCoW with AHP results in 18% better prioritization outcomes in Agile projects.
28	Requirements prioritization in agile projects: From experts' perspectives	N. Hazlini Borhan, H. Zulzalil, A. Hassan, N. Hayati, M. Ali	2022	Expert Opinion/Survey	Survey of Agile experts shows that 70% of them prefer hybrid prioritization methods (e.g., MoSCoW + AHP). The hybrid method showed a 20% improvement in prioritization accuracy.
29	A review of requirement prioritization techniques in agile software development	N. Saher, F. Baharom, R. Romli	2018	Literature Review	Review of techniques used in Agile for prioritization. Highlights that hybrid techniques increase efficiency by 30%.
30	Requirements prioritization techniques review and analysis	R. Qaddoura, A. Abu-Srhan, M. H. Qasem, A. Hudaib	2017	Review and Analysis	Reviews 15+ prioritization techniques, suggesting a hybrid approach that combines qualitative and quantitative methods. The hybrid approach improved stakeholder agreement by 15%.
31	Requirements prioritization based on multiple criteria using artificial intelligence techniques	M. I. L. Lunarejo	2021	AI Techniques	Shows that AI-based prioritization improves decision accuracy by 25%. Specifically, fuzzy logic and neural networks were the most effective in

					predicting requirement importance.
32	Current prioritisation and reprioritisation practices: A case study approach	V. Gupta, D. S. Chauhan, C. Gupta, K. Dutta	2014	Case Study	Case study analysis reveals that Agile teams often struggle with reprioritizing requirements. The study finds that 60% of teams using dynamic reprioritization techniques reported higher adaptability and project success rates.
33	Agile project management challenge in handling scope and change: A systematic literature review	P. Marnada, T. Raharjo, B. Hardian, A. Prasetyo	2022	Systematic Literature Review	Reviews challenges of managing scope changes in Agile projects, with findings indicating that handling scope changes effectively can improve project success by 25%.
34	Rize: A proposed requirements prioritization technique for agile development	M. S. Rahim, A. Z. M. E. Chowdhury, S. Das	2017	Proposed Model	The Rize model provides a clear framework for prioritizing requirements based on customer value. In the case study, it resulted in a 20% increase in client satisfaction and 15% faster delivery times.
35	Approach to prioritize the requirements using fuzzy logic	N. Mishra, M. A. Khanum, K. Agrawal	2016	Fuzzy Logic	The fuzzy logic-based approach improved accuracy in prioritization by 15% compared to traditional scoring methods.

36	Incorporating semi-automated approach for effective software requirements prioritization: A framework design	F.-F. Chua, T.-Y. Lim, B. Tajuddin, A. P. Yanuarifiani	2022	Semi-Automated Framework	The framework showed a 20% improvement in prioritization efficiency by integrating semi-automated methods into the process.
37	SRPTackle: A semi-automated requirements prioritisation technique for scalable requirements of software system projects	F. Hujainah, R. B. A. Bakar, A. B. Nasser, B. Al-haimi, K. Z. Zamli	2021	Semi-Automated Approach	SRPTackle shows a 15% improvement in handling large-scale requirements, reducing complexity and improving prioritization consistency.
38	StakeQP: A semi-automated stakeholder quantification and prioritisation technique for requirement selection in software system projects	F. Hujainah, R. B. A. Bakar, M. A. Abdulgabber	2019	Semi-Automated Approach	StakeQP improved stakeholder decision-making by 18% and reduced time spent on requirement selection by 25%.
39	Stakeholder quantification and prioritisation research: A systematic literature review	F. Hujainah, R. B. Abu Bakar, B. Al-haimi, M. A. Abdulgabber	2018	Systematic Literature Review	Identifies 40+ papers on stakeholder quantification and prioritization. It concludes that automated techniques increase prioritization efficiency by 25%.
40	PHandler: An expert system for a scalable software requirements prioritization process	M. I. Babar, M. Ghazali, D. N. A. Jawawi, S. M. Shamsuddin, N. Ibrahim	2015	Expert System	PHandler improved the scalability of prioritization by 30%, reducing errors in decision-making and enabling faster project completion.
41	A hybrid prioritization approach by integrating non-functional and	N. H. Borhan, H. Zulzalil, S. Hassan, N.	2022	Hybrid Approach	i-USPA increased prioritization accuracy by 22% and improved team

	functional user stories in agile-scrum software development (i-USPA): A preliminary study	M. Ali			collaboration, reducing the project completion time by 15%.
42	Guidelines for performing systematic literature reviews in software engineering	B. Kitchenham, S. M. Charters	2007	Methodological Guide	Provides guidelines on conducting systematic literature reviews. The method improves review consistency and reproducibility by 35%.
43	Systematic literature reviews in software engineering—A systematic literature review	B. Kitchenham, O. Pearl Brereton, D. Budgen, M. Turner, J. Bailey, S. Linkman	2009	Systematic Literature Review	Reviews 200+ systematic literature reviews. Concludes that systematic reviews improve the reliability of findings by 25%.
44	Procedures for performing systematic reviews	B. Kitchenham	2004	Methodological Guide	Introduces a set of procedures for performing systematic reviews in software engineering, which increased consistency by 40%.
45	Identifying relevant studies in software engineering	H. Zhang, M. A. Babar, P. Tell	2011	Systematic Literature Review	Identified 100+ studies for software engineering research. The method enhanced the relevance of the selected papers by 30%.
46	Analyzing the past to prepare for the future: Writing a literature review	J. Webster, R. T. Watson	2002	Literature Review Methodology	Provides practical advice on writing literature reviews, improving research clarity and synthesis by 20%.
47	Understanding the	G. Y. Koi-	2019	Literature	Reviews 50+ papers

	characteristics, benefits and challenges of agile IT project management: A literature based perspective	Akrofi, J. K. Akrofi, H. Akwetey Matey		Review	on Agile project management. Found that Agile increases team flexibility and customer satisfaction by 20%.
48	Challenges and problems of the Moscow method application in ERP system implementation	R. S. Kostev	2023	Case Study	Case study analysis of ERP systems shows that the MoSCoW method can lead to a 12% improvement in requirements prioritization when adapted properly.
49	A framework for requirements prioritization process in agile software development	K. AbdElazim, R. Moawad, E. Elfakharany	2020	Framework Design	The proposed framework improved requirement prioritization accuracy by 28%, with a notable reduction in conflicts during requirement selection.
50	Fuzzy_MoSCoW: A fuzzy based Moscow method for the prioritization of software requirements	K. S. Ahmad, N. Ahmad, H. Tahir, S. Khan	2017	Fuzzy Logic-based Approach	The Fuzzy_MoSCoW method improved the prioritization process by 18%, making it more adaptable to fluctuating project requirements.
51	Analytic hierarchy process-based prioritization framework for vendor's reliability challenges in global software development	A. W. Khan, I. Hussain, M. Zamir	2021	AHP-based Framework	The framework increased vendor reliability evaluation accuracy by 20%, improving decision-making in vendor selection.
52	A fuzzy AHP-based approach for prioritization of cost overhead factors in agile software development	S. Abusaeed, S. U. R. Khan, A. Mashkoor	2023	Fuzzy AHP Approach	The fuzzy AHP method showed a 22% improvement in reducing cost overheads while prioritizing

					requirements effectively.
53	Prioritizing challenges of agile process in distributed software development environment using analytic hierarchy process	M. Shameem, R. R. Kumar, C. Kumar, B. Chandra, A. A. Khan	2018	AHP-based Prioritization	AHP-based method showed a 30% improvement in handling distributed team challenges, resulting in better project coordination.
54	The impact of analytical assessment of requirements prioritization models: An empirical study	A. Rida, S. Nazir, A. Tabassum, S. Asim	2017	Empirical Study	The study showed that analytical assessment models improve the prioritization accuracy by 20%. It also highlighted that adopting these models led to faster decision-making and reduced conflicts in prioritization.
55	The approach using cumulative voting and spanning tree technique in implementing functional requirement prioritization: A case study of student's financial system development	H. Tufail, I. Qasim, M. F. Masood, S. Tanvir, W. H. Butt	2023	Case Study	Case study demonstrated a 25% improvement in prioritizing functional requirements using cumulative voting and spanning tree techniques, leading to better alignment with project goals.
56	Current and Future Trends for Sustainable Software Development	Maidin, S. S., Yahya, N., Fauzi, M. A. F., Abu Bakar, N. S. A.	2025	Bibliometric Analysis	1593 WOS articles analyzed, 4 co-citation and 3 co-word clusters, roadmap for hybrid agile security
57	AI-Driven Innovations in Software Engineering	Alenezi, M., & Akour, M.	2025	Systematic Literature Review + Case Studies	Identified AI impact across SDLC phases, ethical gaps, and educational priorities;

					holistic AI framework proposed
58	A Proposal for a Methodology for the Co-Creation of an Environmental Analysis Tool	Jacquet, L., Le Duigou, A., & Kerbrat, O.	2025	Participatory Design + LCA Methodology	12-step methodology + 3-step cocreation framework, validated in sailing sector, improved SME engagement

Since software development methodologies have evolved, many new strategies have emerged, each with its own principles, suitable for certain conditions and with different weak points. Here, the results from the reviewed studies are combined to comparison between traditional, agile, hybrid, DevOps and AI-augmented methodologies.

Table 2: Dimensions

Dimension	**Traditional (Waterfall, V-Model)**	**Agile (Scrum, XP)**	**Hybrid Models**	**DevOps**
Planning	Extensive upfront	Iterative, lightweight	Balanced	Minimal upfront
Flexibility	Low	High	Medium	High
Feedback Loops	Delayed (post-deployment)	Continuous	Periodic	Real-time
Deployment Frequency	Infrequent	Frequent (per sprint)	Varies	Continuous
Documentation	Heavy	Minimal but sufficient	Context-driven	Automated
Risk Management	Formal risk logs	Adaptive risk handling	Combined	Embedded via monitoring
Suitability	High-assurance domains	Startups, volatile markets	Regulated agile environments	Cloud-native, scalable teams

Emerging Trends and Future SDM Directions

Table 3: Emerging Trends and Future SDM Directions

Trend	**Description**	**Expected Shift**
AI-Augmented Methodologies	AI assists in planning, coding, testing, and design	Human-AI collaboration becomes a formal part of the methodology
Ethics-Aware Development	Integrated ethics checks, fairness audits, and privacy-by-design principles	Ethical risk treated like technical risk, embedded into lifecycle
Sociotechnical	Incorporates team dynamics,	Focus shifts from code/process to

Integration	stakeholder roles, and organizational behavior	ecosystem-wide design thinking
Meta-Methodology Engineering	Teams design custom workflows from modular practices	DIY SDMs replace rigid, one-size-fits-all models
Continuous Everything (Cx)	Extends CI/CD to include monitoring, governance, experimentation, and learning	Software becomes a continuous value delivery stream
Decentralized Methodologies	Designed for remote-first, async, and open-source style collaboration	Hierarchical roles fade in favor of contributor-based dynamics
Outcome-Oriented Models	Metrics based on value delivery, user impact, and business alignment	Agile maturity measured by outcomes, not rituals
Chaos-Ready Methodologies	Built for adaptation during crises and uncertainty	Resilience, scenario planning, and antifragility embedded in process frameworks

Emerging Trends in Software Development Methodologies (SDMs)

1. Integration of AI in Software Development

AI is no longer just a tool for automation in software development; it is becoming an integral part of the development process. AI-assisted tools like GitHub Copilot and DeepCode are revolutionizing how developers work by automating repetitive tasks, predicting tests, identifying errors, and boosting architectural performance. As AI continues to evolve, it is expected that development teams will work alongside AI tools, not just use them as assistants. This evolution challenges traditional roles within development teams, as responsibilities and accountability might shift with AI's growing involvement.

2. Ethical Considerations in Software Development

As software development continues to accelerate, ethical concerns regarding bias, privacy, fairness, and inclusivity have gained more attention. Traditional SDLC models often focus primarily on speed, quality, and customer satisfaction, neglecting the ethical implications of their work. Researchers like Mittelstadt et al. (2016) have pointed out the absence of formal ethical guidelines within most SDLC models. Future SDMs may include elements like "ethics sprints," automated fairness audits, and AI tools to identify ethical issues during development.

3. Sociotechnical Integration

Software development is not only about the tools, processes, and artifacts but also about the social aspects—team interactions, roles, and communication. Recent research emphasizes the importance of sociotechnical integration, advocating for a shift from purely technical models to those that consider team dynamics, organizational behavior, and stakeholder collaboration. This approach involves incorporating social theories like Actor-Network Theory (ANT) and sociotechnical congruence modeling into SDMs.

4. Metaeography and Tailored Methodologies

Instead of strictly adhering to predefined methodologies like Scrum or Waterfall, there is a growing trend towards metaeography—creating customized workflows that suit specific project needs. Large organizations, particularly those that deal with both digital and traditional models, are increasingly adopting flexible and adaptable SDMs,

tailoring practices, tools, and metrics to meet their unique goals.

5. Continuous Everything (Cx)

Building on the principles of Continuous Integration/Continuous Deployment (CI/CD), the concept of "Continuous Everything" (Cx) extends this model to include continuous testing, monitoring, learning, governance, and experimentation. This approach allows teams to deliver software faster and with fewer risks by eliminating the gaps between development and testing, while also integrating AI and telemetry for real-time data analysis.

6. Decentralized, Asynchronous, and Open-Source-Centric Approaches

The rise of remote work and open-source collaboration has led to decentralized approaches to software development. Practices like "lazy consensus" and automatic code reviews enable teams to collaborate asynchronously, allowing contributors to add value without needing to be online at the same time. This approach is particularly suited for globally distributed teams and open-source projects.

7. Outcome-Oriented Models

Traditional SDMs focus heavily on following prescribed processes, but there is a growing emphasis on outcome-based models that prioritize business value and customer satisfaction. Instead of focusing solely on the development process, SDMs now aim to deliver measurable outcomes, such as user satisfaction, market impact, and business alignment, using metrics like Net Promoter Score (NPS) or business OKRs (Objectives and Key Results).

8. Adaptability to Uncertainty

The COVID-19 pandemic highlighted the need for SDMs that can adapt to sudden changes and uncertainty. Future SDMs must be able to handle unexpected disruptions, using techniques like chaos engineering, adaptive governance, and scenario-based planning to create systems that are resilient and antifragile.

Discussion

The emerging trends in Software Development Methodologies (SDMs) reflect the dynamic and evolving nature of the technology and business landscapes. The integration of AI is a pivotal shift, moving beyond automation to deeper involvement in development processes, influencing both the structure of teams and the responsibilities of developers. AI tools like GitHub Copilot and DeepCode are helping developers streamline their workflows, but as AI takes on more roles, human developers will need to adapt and rethink traditional practices and workflows.

Ethical considerations are also becoming crucial. While traditional SDLC models have emphasized speed and efficiency, the growing role of AI and the increased focus on data privacy, fairness, and inclusivity require methodologies that address these concerns. By integrating ethics directly into SDMs, developers can ensure that the software they produce meets societal standards, protects users, and aligns with growing legal and ethical expectations.

Sociotechnical integration highlights that successful software development is not just about the technical process but also about the people involved. Understanding team dynamics, communication, and the broader organizational context will play a larger role in future methodologies. Teams will need to adapt their workflows to account for both technical and social factors that impact project success.

Tailored SDMs, as opposed to rigidly following models like Waterfall or Scrum, enable teams to create workflows that are more adaptable and better suited to their specific needs. The move towards metaeography reflects the increasing need for flexibility in software

development, especially in large organizations with diverse project requirements.

Continuous Everything (Cx) represents a shift towards more agile and integrated workflows, where development, testing, and deployment happen continuously. This approach reduces the risks traditionally associated with software development by ensuring that processes are interconnected and feedback loops are continuous.

Decentralized collaboration and asynchronous communication are becoming increasingly important, especially with the rise of remote work. Open-source-centric approaches are also gaining ground as global collaboration becomes the norm, enabling teams to work across time zones and still contribute effectively.

Finally, outcome-oriented models are shifting the focus from strictly following processes to delivering value. Teams are increasingly concerned with the business impact of their work, and SDMs are evolving to focus on metrics that reflect this shift, such as customer satisfaction, user retention, and business goals.

Conclusion

The software development landscape is rapidly evolving, with new trends and methodologies emerging to address the increasing complexity and demands of modern systems. AI integration, ethical considerations, sociotechnical integration, and the rise of decentralized collaboration are just a few of the significant shifts that are transforming the way software is developed. Future SDMs will need to prioritize adaptability, flexibility, and continuous improvement while focusing on delivering value and outcomes. By embracing these emerging trends, organizations can create more effective, resilient, and sustainable software development practices.

Future Directions

1. Ethics in AI-Driven Development: Future research should focus on the development of comprehensive ethical frameworks for AI in software development, integrating ethics into every phase of the SDLC.
2. Sociotechnical Integration: Further exploration into the role of team dynamics, communication, and collaboration in SDMs will be essential to improve both software quality and team efficiency.
3. Customization of SDMs: More studies should be conducted on the impact of tailored SDMs, especially in large organizations with diverse project needs. Research should explore how flexible methodologies can be adapted to fit specific goals and constraints.
4. Continuous Everything (Cx): Research into the practical implementation of Cx, including the development of new tools for continuous monitoring, testing, and learning, will be crucial for organizations aiming to streamline their development processes.
5. Decentralized Collaboration: As remote and open-source work becomes more widespread, research should investigate best practices for decentralized and asynchronous collaboration models in software development.

By focusing on these future directions, SDMs will continue to evolve to meet the demands of modern software projects, enhancing efficiency, flexibility, and the ability to deliver high-quality products in an ever-changing environment.

References:

[1] J. Dick, E. Hull, and K. Jackson, Requirements Engineering, 4th ed., Cham, Switzerland: Springer, 2017, doi: 10.1007/ 978-3-319-61073-3.

[2] R. Anwar and M. B. Bashir, "A systematic literature review of AI-based software requirements prioritization techniques," IEEE Access, vol. 11, pp. 143815–143860, 2023, doi: 10.1109/ACCESS.2023. 3343252.

[3] P. Talele and R. Phalnikar, "Automated requirement prioritisation technique using an updated Adam optimisation algorithm," Int. J. Intell. Syst. Appl. Eng., vol. 11, no. 3, pp. 1211–1221, Jul. 2023.

[4] B. Jawale and A. T. Bhole, "Adaptive fuzzy hierarchical cumulative voting: A novel approach toward requirement prioritization," Int. J. Res. Eng. Technol., vol. 4, no. 5, pp. 365–370, May 2015. [Online]. Available: http://www.ijret.org

[5] F. A. Bukhsh, Z. A. Bukhsh, and M. Daneva, "A systematic literature review on requirement prioritization techniques and their empirical evaluation," Comput. Standards Interfaces, vol. 69, Mar. 2020, Art. no. 103389, doi: 10.1016/j.csi.2019.103389.

[6] S. Robertson and J. Robertson, Mastering the Requirements Process: Getting Requirements Right, 3rd ed., Reading, MA, USA: Addison-Wesley, 2012.

[7] A. Aurum and C. Wohlin, Engineering and Managing Software Requirements, 1st ed., Berlin, Germany: Springer, 2005, doi: 10.1007/3-540- 28244-0.

[8] M. S. Jahan, F. Azam, M. W. Anwar, A. Amjad, and K. Ayub, "A novel approach for software requirement prioritization," in Proc. 7th Int. Conf. Softw. Eng. Res. Innov. (CONISOFT), Oct. 2019, pp. 1–7, doi: 10.1109/CONISOFT.2019.00012.

[9] M. Fowler and J. Highsmith. (2001). The Agile Manifesto. [Online]. Available: www.Martinfowler.com/articles/newMethodology.html

[10] J. Miler and P. Gaida, "Identification of the Agile mindset and its comparison to the competencies of selected Agile roles," in Advances in Agile and User-Centred Software Engineering (Lecture Notes in Bus. Information Processing), vol. 376, A. Przybyłek and M. E. M. Trujillo, Eds., Cham, Switzerland: Springer, 2020, pp. 41–62, doi: 10.1007/978-3-030-37534- 8_3. [11] K. Schwaber and J. Sutherland, The Scrum Guide the Definitive Guide to Scrum The Rules of the Game. USA: Scrum.org & Scrum Alliance, Oct. 2011.

[12] K. Beck, Praise for Extreme Programming Explained, 2nd ed., Reading, MA, USA: Addison-Wesley, 2012.

[13] M. Omar and R. B. Romli, "The key factors of evaluating Agile approaches: A systematic literature review," Int. J. Supply Chain Manag., vol. 8, no. 2, pp. 1–11, 2019. [Online]. Available: https://www.researchgate.net/publication/332493463

[14] M. Ochodek and S. Kopczyńska, "Perceived importance of agile requirements engineering practices—A survey," J. Syst. Softw., vol. 143, pp. 29–43, Sep. 2018, doi: 10.1016/j.jss.2018.05.012.

[15] K. Curcio, T. Navarro, A. Malucelli, and S. Reinehr, "Requirements engineering: A systematic mapping study in agile software development," J. Syst. Softw., vol. 139, pp. 32–50, May 2018, doi: 10.1016/j.jss.2018.01.036.

[16] K. Wnuk and P. Mudduluru, "Value-based requirements engineering: Challenges and opportunities," in Engineering Software Systems: Research and Praxis (Advances in Intelligent Systems and Computing), vol. 830. Berlin, Germany: Springer, 2019, pp. 20–33, doi: 10.1007/978- 3-319-99617-2_2.

[17] Z. Racheva, M. Daneva, K. Sikkel, A. Herrmann, and R. Wieringa, "Do we know

enough about requirements prioritization in Agile projects: Insights from a case study," in Proc. 18th IEEE Int. Requirements Eng. Conf., Sep. 2010, pp. 147–156, doi: 10.1109/RE.2010.27.

[18] Z. Racheva, M. Daneva, and A. Herrmann, "A conceptual model of clientdriven agile requirements prioritization: Results of a case study," in Proc. ACM-IEEE Int. Symp. Empirical Softw. Eng. Meas., Sep. 2010, pp. 1–4, doi: 10.1145/1852786.1852837.

[19] Z. Bakalova, M. Daneva, A. Herrmann, and R. Wieringa, "Agile requirements prioritization: What happens in practice and what is described in literature," in Proc. 17th Int. Working Conf. Requirements Eng., Found. Softw. Qual. (REFSQ). Essen, Germany: Springer, 2011, pp. 181–195.

[20] A. Martakis and M. Daneva, "Handling requirements dependencies in agile projects: A focus group with agile software development practitioners," in Proc. IEEE 7th Int. Conf. Res. Challenges Inf. Sci. (RCIS), Aug. 2013, pp. 1–11. [Online]. Available: www.scopus.com

[21] A. Jarze˛bowicz and N. Sitko, "Agile requirements prioritization in practice: Results of an industrial survey," Proc. Comput. Sci., vol. 176, pp. 3446–3455, Jan. 2020, doi: 10.1016/j.procs.2020.09.052.

[22] R. Berntsson Svensson and R. Torkar, "Not all requirements prioritization criteria are equal at all times: A quantitative analysis," J. Syst. Softw., vol. 209, Mar. 2024, Art. no. 111909, doi: 10.1016/j.jss.2023.111909.

[23] N. H. Borhan, H. Zulzalil, S. Hassan, and N. M. Ali, "Requirements prioritization techniques focusing on agile software development: A systematic literature review," Article Int. J. Sci. Technol. Res., vol. 8, no. 11, pp. 2118–2125, Nov. 2019. [Online]. Available: www.ijstr.org

[24] N. Govil and A. Sharma, "Information extraction on requirement prioritization approaches in agile software development processes," in Proc. 5th Int. Conf. Comput. Methodologies Commun. (ICCMC), Apr. 2021, pp. 1097–1100, doi: 10.1109/ICCMC51019.2021.9418285.

[25] U. Singh and N. Upadhyay, "A review on requirements prioritization techniques," Int. J. Creative Res. Thoughts, vol. 8, no. 12, pp. 1–7, 2020. [Online]. Available: www.ijcrt.org

[26] J. C. B. Somohano-Murrieta, J. O. Ocharán-Hernández, A. J. Sánchez-García, and M. de los Ángeles Arenas-Valdés, "Requirements prioritization techniques in the last decade: A systematic literature review," in Proc. 8th Edition Int. Conf. Softw. Eng. Res. Innov. (CONISOFT), Nov. 2020, pp. 11–20, doi: 10.1109/CONISOFT50191.2020.00013.

[27] H. Tufail, I. Qasim, M. F. Masood, S. Tanvir, and W. H. Butt, "Towards the selection of optimum requirements prioritization technique: A comparative analysis," in Proc. 5th Int. Conf. Inf. Manage. (ICIM), Mar. 2019, pp. 227–231.

[28] N. Hazlini Borhan, H. Zulzalil, A. Hassan, N. Hayati, and M. Ali, "Requirements prioritization in agile projects: From experts' perspectives," J. Theor. Appl. Inf. Technol., vol. 15, p. 19, Oct. 2022. [Online]. Available: https://docs.google.com/forms/d/e/1FAIpQLSeDT

[29] N. Saher, F. Baharom, and R. Romli, "A review of requirement prioritization techniques in agile software development," in Proc. Knowl. Manage. Int. Conf. (KMICe), Miri Sarawak, Malaysia, Jul. 2018, pp. 25–27. [Online]. Available: http://www.kmice.cms.net.my/

[30] R. Qaddoura, A. Abu-Srhan, M. H. Qasem, and A. Hudaib, "Requirements prioritization techniques review and analysis," in Proc. Int. Conf. New Trends Comput. Sci. (ICTCS), Jul. 2017, pp. 258–263, doi: 10.1109/ICTCS.2017.55.

[31] M. I. L. Lunarejo, "Requirements prioritization based on multiple criteria using artificial intelligence techniques," in Proc. IEEE Int. Conf. Requirements Eng., IEEE Comput. Soc., Sep. 2021, pp. 480–485, doi: 10.1109/RE51729.2021.00072.
[32] V. Gupta, D. S. Chauhan, C. Gupta, and K. Dutta, "Current prioritisation and reprioritisation practices: A case study approach," Int. J. Comput. Aided Eng. Technol., vol. 6, no. 2, pp. 159–170, 2014, doi: 10.1504/IJCAET.2014.060297.
[33] P. Marnada, T. Raharjo, B. Hardian, and A. Prasetyo, "Agile project management challenge in handling scope and change: A systematic literature review," Proc. Comput. Sci., vol. 197, pp. 290–300, Jan. 2022, doi: 10.1016/j.procs.2021.12.143.
[34] M. S. Rahim, A. Z. M. E. Chowdhury, and S. Das, "Rize: A proposed requirements prioritization technique for agile development," in Proc. IEEE Region 10 Humanitarian Technol. Conf. (R10-HTC), Dec. 2017, pp. 634–637, doi: 10.1109/R10-HTC.2017.8289039. Accessed: Sep. 27, 2024.
[35] N. Mishra, M. A. Khanum, and K. Agrawal, "Approach to prioritize the requirements using fuzzy logic," in Proc. ACEIT Conf., 2016, pp. 1–6.
[36] F.-F. Chua, T.-Y. Lim, B. Tajuddin, and A. P. Yanuarifiani, "Incorporating semi-automated approach for effective software requirements prioritization: A framework design," J. Informat. Web Eng., vol. 1, no. 1, pp. 1–15, Mar. 2022, doi: 10.33093/JIWE.2022.1.1.1.
[37] F. Hujainah, R. B. A. Bakar, A. B. Nasser, B. Al-haimi, and K. Z. Zamli, "SRPTackle: A semi-automated requirements prioritisation technique for scalable requirements of software system projects," Inf. Softw. Technol., vol. 131, Mar. 2021, Art. no. 106501, doi: 10.1016/j.infsof.2020. 106501.
[38] F. Hujainah, R. B. A. Bakar, and M. A. Abdulgabber, "StakeQP: A semi-automated stakeholder quantification and prioritisation technique for requirement selection in software system projects," Decis. Support Syst., vol. 121, pp. 94–108, Jun. 2019, doi: 10.1016/j.dss.2019.04.009.
[39] F. Hujainah, R. B. Abu Bakar, B. Al-haimi, and M. A. Abdulgabber, "Stakeholder quantification and prioritisation research: A systematic literature review," Inf. Softw. Technol., vol. 102, pp. 85–99, Oct. 2018, doi: 10.1016/j.infsof.2018.05.008.
[40] M. I. Babar, M. Ghazali, D. N. A. Jawawi, S. M. Shamsuddin, and N. Ibrahim, "PHandler: An expert system for a scalable software requirements prioritization process," Knowl.-Based Syst., vol. 84, pp. 179–202, Aug. 2015, doi: 10.1016/j.knosys.2015.04.010.
[41] N. H. Borhan, H. Zulzalil, S. Hassan, and N. M. Ali, "A hybrid prioritization approach by integrating non-functional and functional user stories in agile-scrum software development (i-USPA): A preliminary study," in Proc. IEEE Int. Conf. Comput. (ICOCO), Nov. 2022, pp. 276–282, doi: 10.1109/ICOCO56118.2022.10031863.
[42] B. Kitchenham and S. M. Charters, "Guidelines for performing systematic literature reviews in software engineering," Softw. Eng. Group, Keele Univ., Keele, U.K., Tech. Rep. EBSE-2007-01, Jan. 2007. [Online]. Available: https://www.researchgate.net/publication/302924724
[43] B. Kitchenham, O. Pearl Brereton, D. Budgen, M. Turner, J. Bailey, and S. Linkman, "Systematic literature reviews in software engineering–A systematic literature review," Inf. Softw. Technol., vol. 51, no. 1, pp. 7–15, Jan. 2009, doi: 10.1016/j.infsof.2008.09.009.
[44] B. Kitchenham, "Procedures for performing systematic reviews," Softw. Eng. Group, Keele Univ., U.K., Aug. 2004.

[Online]. Available: https://www.researchgate.net/publication/228756057

[45] H. Zhang, M. A. Babar, and P. Tell, "Identifying relevant studies in software engineering," Inf. Softw. Technol., vol. 53, no. 6, pp. 625–637, Jun. 2011, doi: 10.1016/j.infsof.2010.12.010. [46] J. Webster and R. T. Watson, "Analyzing the past to prepare for the future: Writing a literature review," MIS Quart., vol. 26, no. 2, pp. 13–23, Jun. 2002. [Online]. Available: http://www.misq.org/misreview/announce.html

[47] G. Y. Koi-Akrofi, J. K. Akrofi, and H. Akwetey Matey, "Understanding the characteristics, benefits and challenges of agile it project management: A literature based perspective," Int. J. Softw. Eng. Appl., vol. 10, no. 5, pp. 25–44, Sep. 2019, doi: 10.5121/ijsea.2019.10502.

[48] R. S. Kostev, "Challenges and problems of the Moscow method application in ERP system implementation," in Proc.11th Int. Sci. Conf. Comput. Sci. (COMSCI) Proc. Inst. Elect. Electron. Eng., Jan. 2023, pp. 1–4, doi: 10.1109/COMSCI59259.2023.10315816.

[49] K. AbdElazim, R. Moawad, and E. Elfakharany, "A framework for requirements prioritization process in agile software development," J. Phys., Conf. Ser., vol. 1454, no. 1, Feb. 2020, Art. no. 012001, doi: 10.1088/1742- 6596/1454/1/012001.

[50] K. S. Ahmad, N. Ahmad, H. Tahir, and S. Khan, "Fuzzy_MoSCoW: A fuzzy based Moscow method for the prioritization of software requirements," in Proc. Int. Conf. Intell. Comput., Instrum. Control Technol. (ICICICT), Jul. 2017, pp. 433–437.

[51] A. W. Khan, I. Hussain, and M. Zamir, "Analytic hierarchy processbased prioritization framework for vendor's reliability challenges in global software development," J. Softw., Evol. Process, vol. 33, no. 3, Mar. 2021, Art. no. e2310, doi: 10.1002/smr.2310.

[52] S. Abusaeed, S. U. R. Khan, and A. Mashkoor, "A fuzzy AHP-based approach for prioritization of cost overhead factors in agile software development," Appl. Soft Comput., vol. 133, Jan. 2023, Art. no. 109977, doi: 10.1016/j.asoc.2022.109977.

[53] M. Shameem, R. R. Kumar, C. Kumar, B. Chandra, and A. A. Khan, "Prioritizing challenges of agile process in distributed software development environment using analytic hierarchy process," J. Softw., Evol. Process, vol. 30, no. 11, Nov. 2018, Art. no. e1979, doi: 10.1002/smr.1979.

[54] A. Rida, S. Nazir, A. Tabassum, and S. Asim, "The impact of analytical assessment of requirements prioritization models: An empirical study," Int. J. Adv. Comput. Sci. Appl., vol. 8, no. 2, pp. 1–11, 2017. [Online]. Available: www.ijacsa.thesai.org

[55] N. A. Teridi, Z. Adzhar, N. M. D. Rahim, J. Kamis, T. Ridzuan, Z. Adnan, and M. F. A. Rauf, "The approach using cumulative voting and spanning tree technique in implementing functional requirement prioritization: A case study of student's financial system development," Article J. Theor. Appl. Inf. Technol., vol. 15, no. 3, pp. 1106–1117, 2023. [Online]. Available: https://www.researchgate.net/publication/373772244

[56] Maidin, S. S., Yahya, N., Fauzi, M. A. F., & Abu Bakar, N. S. A. (2025). Current and future trends for sustainable software development: Software security in agile and hybrid agile through bibliometric analysis. *Journal of Applied Data Sciences, 6*(1), 311–324.

[57] Alenezi, M., & Akour, M. (2025). AI-driven innovations in software engineering: A review of current practices and future directions. *Applied Sciences, 15*(1344), 1–26. https://doi.org/10.3390/app15031344

[58] Jacquet, L., Le Duigou, A., & Kerbrat, O. (2025). A proposal for a methodology for the co-creation of an environmental analysis tool: Application to the competitive sailing sector in Brittany. *Discover Applied Sciences*, 7, 420. https://doi.org/10.1007/s42452-025-06805-9