Transactions on Engineering and Computing Sciences - Vol. 13, No. 02
Publication Date: April 25, 2025
DOI:10.14738/tmlai.1302.18659.
Falah, B., Abufardeh, S., & Tawose, O. (2025). Evaluating Agile Methodologies for Software Requirements and Construction: A SWEBOK-Based Analysis. Transactions on Engineering and Computing Sciences, 13(02). 174-194.

Services for Science and Education

Evaluating Agile Methodologies for Software Requirements and Construction: A SWEBOK-Based Analysis

Bouchaib Falah
Computer Science Department,
Lincoln University, PA, USA

Sameer Abufardeh
Computer, Electrical, and Software Engineering
Department, Embry–Riddle Aeronautical University,
Prescott, AZ, United States

Olamide Tawose
Computer Science Department,
Lincoln University, PA, USA

ABSTRACT

Agile methodologies have gained widespread adoption in software development due to their flexibility, iterative approach, and focus on collaboration. However, their effectiveness in addressing critical software engineering domains, particularly the Software Requirements and Software Construction Knowledge Areas (KAs), remains an open question. This study systematically evaluates five agile methodologies—Extreme Programming (XP), Scrum, Feature-Driven Development (FDD), Rapid Application Development (RAD), and Kanban—using a comparative framework built upon key attributes from the Software Engineering Body of Knowledge (SWEBOK). The assessment considers how each methodology approaches requirements gathering, change management, software design, coding practices, and overall project adaptability. By analyzing these methodologies through the lens of SWEBOK Knowledge Areas, this research aims to identify their strengths, limitations, and applicability to different types of software projects. The findings provide valuable insights for software practitioners, project managers, and educators in selecting the most appropriate agile methodology based on project complexity, team structure, and development constraints. Additionally, this study highlights potential gaps in agile methodologies concerning software engineering best practices, paving the way for future research and methodological enhancements. Ultimately, this research contributes to a deeper understanding of how agile practices align with formal software engineering principles, aiding organizations in making more informed decisions when adopting agile frameworks.

Keywords: Agile Manifesto, Agile Frameworks, Agile Methods, Conceptual Study, Knowledge Areas, SWEBOK.

INTRODUCTION

Software engineering is an engineering discipline encompassing all facets of software production, including software specification, development, validation, and maintenance. It applies engineering principles and methods to create secure, safe, reliable, and efficient software. It also emphasizes practices like version control, documentation, and collaboration, especially for large software developed within teams. It is an everexpanding and dynamic field that continuously evolves to meet the increasing demands for digitalization and software solutions in many industries, making it significant in all facets of daily life. This significance is particularly accentuated by the fact that all developed nations' economies rely heavily on software solutions, thereby driving its prominence.

Today's software industry demands adaptive and flexible software development approaches that can accommodate continuous changes in requirements and deliver results in a timely and efficient manner. Due to the ever-changing and fast-paced nature of today's environments and the demand for large-scale production, the traditional software development process models have failed to keep pace with the new demands. This gap led to the emergence of Agile methodologies with manifestos, which emphasize individuals and interactions over processes and tools, working software over comprehensive documentation, customer collaboration over contract negotiation, and responding to change over a rigid plan. Agile leads the main software development activities continuously and iteratively, accepts changes at any stage, and adds up increments of progress.

The advantage of using agile methods lies in their ability to embrace and adapt to change while producing functional software. However, each method varies in how it defines the software development process. In this paper, the focus will be on comparing and elaborating on the following Agile methods: Scrum, Extreme Programming (XP), Feature-driven development (FDD), Rapid Application Development (RAD), and Kanban. While these methodologies vary in their unique approaches to software development, they are all rooted in the core principles of agility, aiming to deliver working software promptly and ensure customer satisfaction. Thus, the objective is to comprehensively assess each framework's strengths, aiming to enhance the overall software engineering process.

RELATED WORK

Numerous researchers have explored the effectiveness of various project management methodologies in adapting to modern software development needs. For instance, Brych V. and Peryt I. argue that in an era marked by global and constant changes, the most effective approach is the adoption of flexible methodologies [12]. In line with this, Abrahamsson P., Salo O., Ronkainen J., and Warsta J. highlight that the agile model allows for dynamic scheduling, real-time modifications during implementation, continuous improvement, and adaptive response to evolving requirements and newly understood challenges [13]. These studies support the growing consensus that flexibility and responsiveness are critical success factors in today's software projects, reinforcing the need for methodologies like Agile. In his position paper, Parsons analyzes the limited coverage of Agile and Lean methodologies in SWEBOK Version 3, highlighting a disconnect between contemporary industry practices and the educational framework provided by SWEBOK. He advocates for a comprehensive revision incorporating

Agile and Lean approaches, emphasizing their prevalence in modern software engineering [14]. In their research, Saleh M. Al-Saleem and Hanif Ullah [1] presented and compared Extreme Programming, Rational Unified Process, and Dynamic System Development Method. They used the Multi-Criterion Decision Making tool to evaluate these methodologies by focusing on processes, practices, advantages, and disadvantages. However, their comparison did not include Scrum, Rapid Application Development (RAD), Kanban, or Feature-driven Development (FDD).

In their paper, João M. Fernandez and Mauro Almeida [2] focused on comparing and classifying agile methods using select attributes related to the knowledge domains of the IEEE Software Engineering Knowledge Association (SWEBOK) and agile manifesto principles. Their technique aims to evaluate the coverage of considered KAs (Knowledge Areas) and the agility of each method, such as comparing Extreme Programming (XP) and Scrum. However, they analyzed these methods primarily based on particular characteristics and practices. At the same time, they could have tackled a wider range of agile methods by comparing their processes and practices with the pros and cons. Additionally, their approaches were more quantitative than qualitative, while a more qualitative parametric scale could improve the depth and accuracy of their analysis.

Ashraf Ferdouse Chowdhury's study, conducted together with Mohammad Nazmul Huda [3], supports and expands on the work of João M. Fernandez and Mauro Almeida [2] in comparing agile software development methodologies. While Fernandes and Almeida focused on comparing XP and Scrum, Chowdhury and Huda's work broadened the comparison to include ASD (Adaptive Software Development) and FDD. They utilize specific attributes linked to Software Requirements and Software Construction to add depth to the comparison. Furthermore, Chowdhury and Huda's study incorporates qualitative and quantitative analysis methods, resulting in a more complete evaluation of ASD and FDD. Their explicit classification criteria, such as Not Satisfied (NS), Partially Satisfied (PS), and Adequately Satisfied (AS), make the comparison results more understandable. As a result, Chowdhury and Huda's work enhances Fernandes and Almeida's methods by integrating more methodologies and a structured evaluation system, leading to a more precise comparison.

Iryna Zasornova et al. [11] present a comparative analysis focused exclusively on Scrum and Kanban, two of the most widely adopted agile methodologies in contemporary software development. Their study highlights the key roles defined in both approaches and examines team events. However, the analysis is limited in scope, as it does not extend the comparison to other agile methodologies that are increasingly relevant in modern software engineering practices.

METHODOLOGY

Given the intricate interplay among various software development practices, directly comparing agile methodologies can be complex. Traditional approaches, which solely emphasize project outcomes, may overlook nuanced distinctions in how each methodology addresses specific aspects of the software development process. Our work adopts a comparative framework built upon key attributes derived from the established body of

knowledge for software engineering, namely the Software Engineering Body of Knowledge (SWEBOK) [4]. This framework enables a more structured and quantifiable comparison.

The methodology employed in this paper follows a multi-step process. Initially, core SWEBOK Knowledge Areas (KAs) directly relevant to agile development will be identified. These KAs include Software Requirements (KA1) and Software Construction (KA4). Subsequently, we will examine each agile methodology (Scrum, Kanban, Extreme Programming (XP), Rapid Application Development (RAD), and Feature-Driven Development (FDD)) through the prism of the chosen SWEBOK KAs. This analysis will evaluate how well each method satisfies the sub-attributes of each KA. After this analysis, the results will be presented by calculating the percentage of satisfaction for each methodology across the sub-attributes within each KA. This approach will provide a nuanced comparison, revealing the strengths and weaknesses of each methodology in addressing the specific details of each KA. Subsequently, we will explore how these distinctions might influence project outcomes (e.g., speed, quality) and discuss the suitability of each method for projects with specific characteristics. We can achieve a more precise and quantifiable comparison of agile methodologies by employing this structured approach, focusing on sub-attribute satisfaction percentages.

Scoring System:

- **NS (Not Satisfied):** The methodology does not adequately address a specific criterion within a KA.
- **PS (Partially Satisfied):** The methodology partially addresses a criterion, but some aspects are missing or underdeveloped.
- **AS (Adequately Satisfied):** The methodology effectively addresses a criterion, demonstrating strong practices that align with the established criteria for the KA.

The scores allocated within the table will play a pivotal role in the conclusive analysis of the results. This analysis aims to identify the strengths and weaknesses of each methodology across various KAs and explore their suitability for projects with specific characteristics. By utilizing this structured approach coupled with a scoring system, we can attain a more refined and measurable comparison of agile methodologies.

CONCEPTUAL FOUNDATIONS OF AGILE METHODS: XP, SCRUM, KANBAN, FDD, AND RAD

This section dives into the fundamental principles of five leading Agile methodologies: Extreme Programming (XP), Scrum, Kanban, Feature Driven Development (FDD), and Rapid Application Development (RAD). Grasping these core concepts will establish a robust foundation for comparison, enabling us to analyze the suitability of each method for various types of software projects.

Extreme Programming (XP)

Extreme Programming (XP) is an Agile software development methodology designed to deliver highquality software that readily adapts to evolving customer needs [5]. This is accomplished by fostering close collaboration between developers and customers, emphasizing clear communication, and implementing iterative development cycles with frequent feedback loops.

Key Practices:

XP prescribes a set of key practices that operationalize its core values and guide the software development process. These practices are outlined in Table 1.

Table 1: Key Practices Of XP

Key Practice	Description
Pair Programming	Two developers work together on a task, one writing code (driver) while the other reviewing (navigator). This promotes knowledge sharing, improves code quality, and fosters code ownership [5].
Test-Driven Development (TDD)	Unit tests are written before writing the actual code. This ensures the code fulfills its intended purpose and promotes maintainability. Tests are continually refined as development progresses.
Continuous Integration (CI)	Developers frequently integrate their code changes into a shared codebase, typically multiple times a day. This allows for early detection and resolution of integration issues.
Continuous Delivery (CD)	(Often considered an extension of CI) The integrated code is automatically deployed to a testing or production environment after passing the automated tests. This enables faster delivery of new features and reduces the risk of deploying buggy code [5].
Planning Game	A collaborative session involving developers, customers, and stakeholders to define user stories, prioritize features, and estimate workload for the next development iteration.
Refactoring	Improving the internal structure of code without changing its functionality. This enhances code readability, maintainability, and long-term maintainability.
Small Releases	XP emphasizes delivering functional software in short iterations (typically 1-2 weeks). This allows for early feedback from customers and facilitates course correction if needed [5].
User Stories	Brief descriptions of features from the customer's perspective, written in a clear and concise manner. These user stories form the basis for planning and estimation in XP.
The Customer	In XP, the customer plays an active role throughout the development process. They collaborate with developers, provide continuous feedback, and prioritize Feature.

By adhering to these core values and implementing these key practices, XP empowers development teams to deliver high-quality software that caters to the evolving needs of their customers.

Scrum

Scrum is a popular Agile framework for managing complex projects and software development. It emphasizes iterative, incremental development that delivers working software in short cycles called sprints [6]. Scrum promotes team collaboration through roles, events, and artifacts that guide the development process [6]. Scrum is built upon a specific framework consisting of roles, events, and artifacts, as depicted in Tables 2, 3, and 4, respectively, which collectively guide the development process.

Table 2: Main Scrum Roles

Role	Description
Product Owner	The Product Owner represents the stakeholders and manages the product backlog, prioritizes features, and ensures the product vision is met.
Scrum Master	The Scrum Master acts as a facilitator and coach, ensuring the Scrum process is followed effectively and removng impediments for the development team.
Development Team	The Development Team is a self-organizing and cross-functional group of individuals responsible for delivering he product backlog items within each Sprint.

Table 3: Main Scrum Events

Event	Description
Sprint Planning	A collaborative session at the beginning of a sprint where the Development Team selects items from the product backlog for the upcoming Sprint and plans how to deliver them.
Daily Scrum	A brief daily meeting (typically 15 minutes) for the Development Team to discuss progress on sprint backlog items, identify any roadblocks, and plan the work for the next day.
Sprint Review	A meeting held at the end of a sprint to showcase the completed product backlog items to stakeholders and receive feedback.
Sprint Retrospective	A meeting held after the sprint review to discuss what went well, what could be improved, and how to adapt the process for future sprints.

Table 4: Main Scrum Artifacts

Artifact	Description
Product Backlog	A prioritized list of features and requirements for the product, continuously refined and updated throughout the development process.
Sprint Backlog	A subset of the product backlog containing items selected for a specific sprint. The Development Team estimates the effort required to complete each item.
Increment	The sum of all product backlog items completed during a sprint, representing a potentially shippable product increment.

By adhering to these core principles and employing the Scrum framework, teams can effectively manage complex projects, deliver working software iteratively, and continually adapt to changing requirements.

Kanban

Kanban is an Agile methodology that visualizes workflow and optimizes work items' flow through a system [7]. Unlike Scrum's structured sprints, Kanban employs a continuous flow model, allowing greater flexibility and adaptation to changing priorities [7].

The cornerstone of Kanban is the Kanban board, which visually represents the workflow stages through which work items progress [7]. Typically, the board consists of columns representing various stages (e.g., To Do, In Progress, Done), with cards representing individual work items moving across the columns as they are completed [7]. While Kanban is less structured than Scrum, it employs several key practices, as outlined in Table 5, to effectively manage workflow.

Table 5: Kanban Key Practices

Key Practice	Description
Limiting Work in Progress (WIP)	Setting limits on the number of tasks allowed in each workflow stage prevents overburdening the team and ensures focus on completing current tasks before starting new ones.
Visualizing Workflow	The Kanban board serves as a central communication hub, allowing everyone to see the overall workflow status, identify bottlenecks, and track progress.
Continuous Flow	Work items are continuously pulled through the workflow based on capacity and priority, promoting efficiency and faster delivery.
Regular Kanban Meetings	The team holds regular meetings to discuss workflow bottlenecks, identify areas for improvement, and review WIP limits

By implementing these core principles and practices, Kanban empowers teams to manage work effectively, adapt to changing priorities, and continually improve their workflow.

Feature Driven Development (FDD)

Feature Driven Development (FDD) is an Agile methodology that emphasizes iterative development with a focus on delivering working features in a timely manner [8]. It combines traditional project management elements with Agile practices to provide a structured yet adaptable approach to software development [8]. FDD follows a five-step development process, as detailed in Table 6, providing a structured framework for delivering features. Additionally, FDD incorporates several key practices to support its process, as outlined in Table 7.

Table 6: Feature-Driven Development Process

Step	Description
Develop an Overall Model	The team establishes a high-level domain model that outlines the core functionalities and entities within the system.
Build a Features List	Working with the customer, the team collaboratively identifies and prioritizes a comprehensive list of features for the entire project.
Plan by Feature	Each Feature is further broken down into smaller, more manageable tasks. The team estimates effort and creates a detailed plan for developing each Feature.
Design by Feature	The team performs detailed design work for each Feature, focusing on technical specifications and implementation details.
Build by Feature	The development team implements the planned features, adhering to design specifications and conducting unit testing throughout the process.

Table 7: Feature Driven Development Key Practices

Practice	Description
Chief Programmer Team	A small, highly skilled development team led by a Chief Programmer is responsible for design and implementation.
Feature Teams (Optional)	For larger projects, temporary feature teams might be formed to focus on specific features, promoting collaboration and ownership.
Daily Stand-up Meetings	The team holds brief daily meetings to discuss progress, identify roadblocks, and ensure everyone is aligned.
Inspections	Regular code inspections are conducted to identify and address potential issues early in the development cycle, promoting quality and maintainability.

FDD presents a compelling option for teams seeking a structured Agile approach. By integrating iterative development with close customer collaboration and quality-focused practices, FDD enables teams to consistently deliver working features and attain predictable results in software development projects.

Rapid Application Development (RAD)

Rapid Application Development (RAD) is an iterative Agile technique that prioritizes speed and user-centered design. Unlike Scrum's organized sprints or Kanban's continuous flow, RAD emphasizes rapid prototyping and ongoing user participation throughout the development lifecycle [9]. This approach ensures that the final software product closely matches user needs and expectations, while its iterative nature allows for course correction and improvement based on ongoing feedback.

RAD's origins can be traced back to the 1980s when James Martin developed a systematic technique centered on rapid prototyping [10]. Over time, RAD evolved to embrace Agile principles, making it a valuable tool for projects requiring quick turnaround times and a high level of user participation. RAD follows a defined set of phases that guide the rapid development process, as detailed in Table 8, and incorporates several key practices, as outlined in Table 9, to support its rapid development approach.

Table 8: Rapid Application Development Phases.

Phase	Description
Business Requirements Planning	Stakeholders and users collaborate to define the project goals, identify functionalities, and establish business requirements for the software.
User Design Workshop	A facilitated session where users actively participate in designing the software's user interface (UI) and user experience (UX) through prototyping and feedback.
Rapid Prototyping	The development team creates functional prototypes that mimic the core functionalities of the software. These prototypes are used to gather user feedback and refine requirements.
Construction	Based on the validated prototypes and user feedback, the development team implements the actual software using reusable components and development tools to expedite the process.
Testing and Turnover	The developed software undergoes rigorous testing to ensure quality and functionality. Once approved, the software is delivered to the users for deployment.

Table 9: Rapid Application Development Key Practices.

Practice	Description
Focus Groups	User groups are actively involved throughout the development process to provide feedback on prototypes and ensure the software meets their needs.
Data Gathering Techniques	Various methods like interviews, surveys, and usability testing are employed to gather user input and refine the software based on their feedback.

Code Reusability	RAD leverages pre-written, reusable code components to expedite development and reduce coding efforts.
Computer-Aided Software Engineering (CASE) Tools	RAD utilizes CASE tools to automate tasks like data modeling, code generation, and documentation, further accelerating the development process.

By prioritizing speed, user-centered design, and iterative development, RAD enables teams to swiftly deliver functional software that aligns with user expectations. However, RAD might not be ideal for projects with complex requirements or extensive post-release maintenance needs.

CLASSIFICATION OF AGILE METHODS USING SWEBOK KAS

Following the conceptual descriptions of various agile methodologies, this section delves into a detailed comparison using the established framework based on Software Engineering Body of Knowledge (SWEBOK) Knowledge Areas (KAs). We will systematically examine each KA through its sub-attributes, allowing for a granular analysis of how well each agile methodology satisfies the sub-attributes within that KA.

Software Requirements KA

Our analysis begins with Software Requirements, a fundamental knowledge area in software engineering. Effective requirements gathering and management are crucial for project success. Here, we delve into the sub-attributes that define how each agile methodology handles this critical aspect of software development. Table 10 summarizes these sub-attributes.

Table 10: Sub-Attributes of Software Requirements Knowledge Area [2]

Sub-Attribute	Description
Software Requirements Fundamentals	Refers to the precise definition of software requirements, distinguishing between different types of requirements (e.g., product vs. system, functional vs. non-functional)
Requirements Process	Refers to the definition of a clear process for the specification, analysis, and validation of requirements, specifying all actors, proposed practices, and management models
Requirements Elicitation	Refers to the way software requirements are elicited and which actors are involved in the process
Requirements Analysis	Refers to the proposed practices for detecting and solving conflicts and for classifying software requirements according to a predefined metric
Requirements Validation	Refers to the proposal of concepts, practices, or techniques to allow the validation of the implemented requirements

Extreme Programming:

Table 11 summarizes how Extreme Programming (XP) addresses the sub-attributes of Software Requirements (KA1). While XP excels in capturing different requirement types (functional vs. non-functional) and fostering close customer collaboration, adequately satisfying both the Fundamentals and Elicitation subattributes, it relies less on a formal requirements process, making it partially satisfy the Process sub-attribute. Although practices like user story refinement and acceptance testing help identify and validate requirements ("PS"

and "AS" respectively), XP might not emphasize formal classification using predefined metrics compared to some other traditional methodologies.

Table 11: XP Assessment in Addressing Sub-Attributes of Software Requirements KA

Sub-Attribute	Classification (Score)	Aspects of XP that Satisfy the Attribute
Software Requirements Fundamentals	AS (Adequately Satisfied)	XP emphasizes user stories, which can inherently capture different requirement types (functional vs. non-functional) through clear descriptions and acceptance criteria.
Requirements Process	PS (Partially Satisfied)	XP focuses on iterative development and user collaboration, but there's less formality in defining a comprehensive process for requirements specification, analysis, and validation compared to some traditional methodologies.
Requirements Elicitation	AS (Adequately Satisfied)	XP promotes close collaboration with customers throughout the development process. User stories are written collaboratively between developers and customers, ensuring a shared understanding of requirements.
Requirements Analysis	PS (Partially Satisfied)	XP practices like user story refinement and acceptance testing help identify and resolve conflicts between requirements. However, there might be less emphasis on the formal classification of requirements using predefined metrics compared to some traditional approaches.
Requirements Validation	AS (Adequately Satisfied)	XP heavily relies on practices like acceptance testing and continuous integration to ensure implemented requirements meet user expectations. Acceptance tests are written collaboratively with customers, ensuring requirements are validated throughout the development process.

Scrum:

Transitioning from evaluating Extreme Programming (XP) in addressing Software Requirements subattributes, we now focus on Scrum's approach. Scrum adequately addresses Software Requirements Fundamentals, thanks to the Product Backlog with user stories that partly differentiate between functional and non-functional needs. However, user story details may vary. Similarly, Scrum satisfactorily handles the Requirements Process with backlog refinement involving stakeholders and the development team. Nevertheless, there's less emphasis on formal, step-by-step requirements definition than other methodologies. Scrum is partially satisfied in Requirements Elicitation because, although product backlog refinement includes stakeholders, the Product Owner holds primary responsibility, potentially limiting input. In the Analysis sub-attribute, Scrum achieves a partially satisfied score with user story refinement during Sprint Planning and backlog refinement to identify and address requirement conflicts but lacks formal classification using predefined metrics. Finally, validation is partially satisfied as user story acceptance criteria guide testing, yet Scrum lacks dedicated practices for ongoing validation throughout Sprint. Table 11 summarizes how Scrum addresses the sub-attributes of the Software Requirements KA.

Table 12: Scrum Assessment in Addressing Sub-Attributes of Software Requirements KA

Sub-Attribute	Classification (Score)	Aspects of Scrum that Satisfy the Attribute
Software Requirements Fundamentals	AS (Adequately Satisfied)	Utilizes a Product Backlog containing user stories to differentiate between functional and non-functional requirements, but varying levels of detail in user stories
Requirements Process	AS (Adequately Satisfied)	Defines a backlog refinement process for clarifying and prioritizing user stories, with less emphasis on a step-by-step process for requirements analysis and validation
Requirements Elicitation	PS (Partially Satisfied)	Involves stakeholders in Product Backlog refinement, but primary responsibility often falls on the Product Owner, limiting the breadth of input.
Requirements Analysis	PS (Partially Satisfied)	Focuses on user story refinement during Sprint Planning, with less emphasis on formal classification using predefined metrics
Requirements Validation	PS (Partially Satisfied)	Utilizes acceptance criteria for user stories in user acceptance testing, lacks dedicated ongoing validation practices throughout the Sprint

Kanban:

Next, we will focus on Kanban's practices and how they deal with the software requirements Knowledge area. Kanban's approach to software requirements prioritizes flexibility and continuous workflow over rigid processes. This impacts how Kanban addresses various sub-attributes of this KA, as summarized in Table 13. In Software Requirements Fundamentals, Kanban achieves and is Adequately Satisfied thanks to visual boards that distinguish requirement types but may lack detailed categorization. Similarly, Kanban Adequately Satisfies the Requirements Process sub-attribute due to its flow-based system that adapts to changing needs, though formalized analysis steps might be absent. Kanban excels in Requirements Elicitation, achieving an Adequately Satisfied score. This can be attributed to stakeholders actively participating through visual management and feedback loops, ensuring ongoing requirement gathering. Analysis receives a Partially Satisfied score. While continuous flow facilitates analysis, Kanban lacks predefined metrics for classifying requirements. Finally, validation is also Partially Satisfied. Kanban uses flow metrics to monitor progress and identify bottlenecks, but dedicated practices for ongoing validation throughout development might be absent.

Table 13: Kanban Assessment in Addressing Sub-Attributes of Software Requirements KA

Sub-Attribute	Classification (Score)	Aspects of Kanban that Satisfy the Attribute
Software Requirements Fundamentals	AS (Adequately Satisfied)	Utilizes visual boards to manage work items, distinguishing between different types of requirements, but may lack detailed categorization

Requirements Process	AS (Adequately Satisfied)	Defines a flow-based process for work items, with continuous improvement and adaptation, but may not have formalized steps for requirements analysis
Requirements Elicitation	AS (Adequately Satisfied)	Involves stakeholders in visual management and continuous feedback loops, facilitating active participation in requirement gathering
Requirements Analysis	PS (Partially Satisfied)	Relies on continuous flow and feedback for analysis but may not have predefined metrics for classifying requirements
Requirements Validation	PS (Partially Satisfied)	Uses flow metrics for validation and continuous improvement but may lack dedicated practices for ongoing validation throughout the process

Feature Driven Development

Feature Driven Development (FDD) adheres to strong principles regarding software requirements. This focus on well-defined processes impacts how FDD addresses various sub-attributes, as summarized in Table 14. Regarding Software Requirements Fundamentals, FDD achieves "Adequately Satisfied" by clearly differentiating between various types of requirements [3]. This distinction ensures a clear understanding of project needs. Similarly, the Requirements Process sub-attribute is Adequately Satisfied in FDD. The methodology defines clear requirements specification, analysis, and validation stages within its overall process [3]. Additionally, FDD identifies key actors and their activities related to requirements [3]. However, FDD receives a "Partially Satisfied" score in Requirement Elicitation. While FDD involves domain experts in high-level requirement reviews, it might lack a more comprehensive approach for gathering and managing requirements across the project lifecycle [3]. On the other hand, FDD shines in Requirement Analysis, achieving "Adequately Satisfied." FDD classifies requirements based on priority and actively reduces conflicts by emphasizing integration, continuous processes, and small releases [3]. This structured approach ensures a well-analyzed set of requirements. Finally, FDD adequately satisfies the validation sub-attribute. Phase 2 of the FDD process focuses on building and validating a prioritized feature list [3]. This ensures each Feature aligns with project goals.

Table 14: FDD Assessment in Addressing Sub-Attributes of Software Requirements KA [2]

Sub-Attribute	Classification (Score)	Aspects of FDD that Satisfy the Attribute
Software Requirements Fundamentals	AS (Adequately Satisfied)	Clearly differentiates between requirement types (functional and non-functional).
Requirements Process	AS (Adequately Satisfied)	Defines stages for specification, analysis, and validation. Identifies key actors and activities.
Requirement Elicitation	PS (Partially Satisfied)	Lacks a comprehensive approach for requirement gathering and management.
Requirement Analysis	AS (Adequately Satisfied)	Classifies requirements and reduces conflicts through prioritization and focus on integration.
Requirement Validation	AS (Adequately Satisfied)	Validates prioritized feature list during phase 2 of the FDD process.

Rapid Application Development:
Rapid Application Development (RAD) embraces a fast-paced approach to software requirements, emphasizing adaptability through rapid prototyping and iterative cycles. This focus on speed impacts how RAD addresses various sub-attributes within the software requirements domain.

While RAD leverages a Product Backlog with user stories, the level of detail within these stories can vary significantly [1, 2]. This inconsistency can make it challenging to clearly distinguish between different types of requirements, making it partially satisfy the Software Requirements Fundamentals.

Similarly, the Requirements Process achieves a Partially Satisfied score. Though collaboration between stakeholders and the development team occurs during backlog refinement, RAD places less emphasis on a rigorously defined, systematic process compared to some methodologies [2]. This fosters flexibility but can lead to a less structured approach to requirement handling.

Stakeholder involvement exists in Requirement Elicitation, but the Product Owner holds primary responsibility. This structure can potentially limit the breadth of input and diverse perspectives on project needs, resulting in a Partially Satisfied score for this sub-attribute.

In Requirement Analysis, RAD fares better, achieving Adequately Satisfied. User story refinement during Sprint Planning helps identify and address potential conflicts between requirements [2]. However, RAD lacks a formal classification system using predefined metrics like other methodologies discussed. Finally, the validation sub-attribute is adequately satisfied. The user story acceptance criteria guide testing at the end of each Sprint, ensuring that features align with expectations. The evaluation of FDD in addressing the subattributes of the Software Requirements KA is summarized in Table 15.

Table 15: RAD Assessment in Addressing Sub-Attributes of Software Requirements KA

Sub-Attribute	Classification (Score)	Aspects of RAD that Satisfy the Attribute
Software Requirements Fundamentals	PS (Partially Satisfied)	User stories may lack details to clearly differentiate requirement types.
Requirements Process	PS (Partially Satisfied)	Backlog refinement with stakeholders, but less emphasis on formal process steps.
Requirement Elicitation	PS (Partially Satisfied)	Stakeholder involvement, but the Product Owner holds primary responsibility.
Requirement Analysis	AS (Adequately Satisfied)	User story refinement during Sprint Planning and backlog refinement facilitates ongoing analysis and conflict resolution.
Requirement Validation	AS (Adequately Satisfied)	User story acceptance criteria guide testing at the end of each Sprint, ensuring features align with expectations.

Software Construction

Following our exploration of Software Requirements (KA1), we now focus on Software Construction (KA4). This KA encompasses the detailed creation of functional software through coding, verification, unit testing, integration testing, and debugging practices. Here, we will examine how each agile methodology addresses these crucial software-building aspects. Table 16 presents the sub-attributes of Software Construction.

Table 16: Sub-Attributes of Software Construction Knowledge Area.

Sub-Attribute	Description
Minimizing Complexity	Techniques and practices used to make the code easier to understand and manage.
Response to Changes	Techniques and practices that make the code easier to modify and adapt to future requirements.
Constructing for Verification	Techniques and practices that allow for easier identification of bugs and errors during development and testing.
Standards in Construction	Following established coding conventions and best practices for consistency and clarity.

Extreme Programming:

Extreme Programming (XP) is characterized by its emphasis on collaboration, rapid feedback, and continuous improvement in software construction. This agility-centric approach significantly influences how XP tackles various sub-attributes in this domain.

While XP places a strong emphasis on enhancing code readability through practices such as pair programming and code reviews, the absence of a strict coding standard poses a potential risk of introducing inconsistencies. Moreover, XP's emphasis on rapid iterations may sometimes prioritize functionality over initial design, potentially affecting long-term maintainability. These considerations collectively contribute to a Partially Satisfied score for Minimizing Complexity.

In terms of Response to Changes, XP excels. Its dedication to continuous testing, refactoring, and a testdriven development (TDD) methodology renders the codebase more adaptable to evolving requirements. Unit tests, constructed with TDD, not only serve as living documentation but also act as regression safety nets, thereby facilitating seamless future modifications. This concerted effort merits XP an Adequately Satisfied score.

XP also secures an Adequately Satisfied score for the Constructing for Verification sub-attribute. The integration of practices like pair programming and continuous integration fosters early bug detection and rectification. Additionally, unit testing with TDD proves instrumental in identifying defects at the code unit level.

Standards in Construction constitute another sub-attribute where XP attains a Partially Satisfied score. Despite promoting code clarity through methodologies like pair programming, XP often lacks a formally defined coding standard. Consequently, this deficiency can result in discrepancies across the codebase, potentially impinging on its maintainability in the long term.

Table 17 encapsulates the assessment of XP in addressing Sub-Attributes of software construction KA.

Table 17: XP Assessment in Addressing Sub-Attributes of Software Construction KA

Sub-Attribute	Classification (Score)	Aspects of XP that Satisfy the Attribute
Minimizing Complexity	PS (Partially Satisfied)	Pair programming and code reviews promote readability, but lack of strict standards can lead to inconsistencies.
Response to Changes	AS (Adequately Satisfied)	Continuous testing, refactoring, and TDD make code adaptable to changes.
Constructing for Verification	AS (Adequately Satisfied)	Pair programming, continuous integration, and TDD unit testing facilitate early bug detection.
Standards in Construction	PS (Partially Satisfied)	Code clarity is emphasized, but formally defined coding standards might be absent.

Scrum:

Scrum prioritizes the delivery of working software in short cycles (Sprints) while emphasizing continuous inspection and adaptation. This iterative approach significantly influences how Scrum addresses various subattributes within software construction.

In terms of Minimizing Complexity, Scrum achieves a Partially Satisfied score. While user stories and backlog refinement foster clear requirements communication, Scrum itself does not prescribe specific coding practices for readability or maintainability. The responsibility for establishing best practices within the Scrum framework lies with the development team.

However, Scrum excels in Response to Changes, earning an Adequately Satisfied score. The fundamental practices of continuous integration and short Sprints facilitate frequent testing and code review, enabling the prompt identification and resolution of changes in requirements throughout the development cycle.

Similarly, the Constructing for Verification sub-attribute gains an Adequately Satisfied score. Scrum's emphasis on continuous integration practices facilitates early bug detection, and the inclusion of user stories and acceptance criteria guides testing efforts, ensuring that features meet expectations. On the other hand, Standards in Construction presents a challenge for Scrum, resulting in a Not Satisfied score. Scrum does not enforce specific coding standards, leaving room for variability across the codebase. Table 18 presents the assessment of Scrum in addressing Sub-Attributes of software construction KA.

Table 18: Scrum Assessment in Addressing Sub-Attributes of Software Construction KA.

Sub-Attribute	Classification (Score)	Aspects of Scrum that Satisfy the Attribute
Minimizing Complexity	PS (Partially Satisfied)	User stories promote clear communication, but specific coding practices for readability are not mandated.
Response to Changes	AS (Adequately Satisfied)	Continuous integration and short Sprints facilitate frequent testing and adaptation to changing requirements.

Constructing for Verification	AS (Adequately Satisfied)	Continuous integration and user story acceptance criteria promote early bug detection.
Standards in Construction	NS (Not Satisfied)	Specific coding standards are not enforced, potentially leading to inconsistencies.

Kanban:

In contrast to structured methodologies, Kanban embraces a visual workflow and a philosophy of continuous improvement in software construction. This emphasis on flexibility and adaptability significantly influences how Kanban addresses various sub-attributes within this domain.

Minimizing Complexity in Kanban receives a Partially Satisfied score. While Kanban encourages clear work item definitions, it does not enforce specific coding practices for readability or long-term maintainability. Consequently, the development team is responsible for establishing best practices within the Kanban framework. Additionally, focusing on rapid flow might prioritize task completion over in-depth code reviews, potentially affecting long-term maintainability.

However, Kanban excels in Response to Changes, achieving an Adequately Satisfied score. The core principle of limiting work in progress (WIP) ensures focus and reduces context switching, allowing developers to adapt more readily to emerging requirements. Furthermore, the visual nature of the Kanban board fosters communication and transparency, facilitating the identification and prioritization of changes.

Similarly, Constructing for Verification garners an Adequately Satisfied score. Kanban's emphasis on continuous flow often leads to adopting automated testing practices, aiding in early bug detection. Additionally, the Kanban board itself, with its focus on visualizing work stages, aids in tracking defect resolution throughout the workflow.

Standards in Construction present a challenge for Kanban, resulting in a Not Satisfied score, as Kanban does not enforce specific coding standards. While teams can adopt best practices and coding conventions, the absence of a mandated standard can lead to inconsistencies across the codebase. Table 19 summarizes the assessment of Kanban in addressing Sub-Attributes of software construction KA.

Table 19: Kanban Assessment in Addressing Sub-Attributes of Software Construction KA.

Sub-Attribute	Classification (Score)	Aspects of Kanban that Satisfy the Attribute
Minimizing Complexity	PS (Partially Satisfied)	Clear work item definition, but specific coding practices for readability are not mandated. Focusing on flow might impact an in-depth code review.
Response to Changes	AS (Adequately Satisfied)	Limiting WIP and visual Kanban board facilitate adaptation to changing requirements.

Constructing for Verification	AS (Adequately Satisfied)	Emphasis on continuous flow often involves automated testing. Kanban board aids in tracking defect resolution.
Standards in Construction	NS (Not Satisfied)	Specific coding standards are not enforced, potentially leading to inconsistencies.

Feature Driven Development:
FDD approaches software requirements with a structured, iterative method that prioritizes manageability and control, significantly influencing how it addresses various sub-attributes. Concerning Problem Complexity, FDD earns an "Adequately Satisfied" rating by systematically decomposing large problems into smaller, more manageable ones. This meticulous breakdown ensures that each component can be effectively addressed and integrated later on.

Similarly, FDD achieves an "Adequately Satisfied" score in Response to Unexpected Changes. This is attributed to its emphasis on continuous integration, refactoring, and collective code ownership. Practices like pair programming and unit testing further bolster FDD's capability to adapt to emerging changes.

In Construction for Verification, FDD excels, earning an "Adequately Satisfied" rating. FDD uses pair programming and unit testing to ensure code quality and facilitates early issue detection, as detailed in Table 20. However, FDD's approach to Standards in Construction falls short, resulting in a Partially Satisfied rating. While FDD stresses standardization during the implementation phase, it may not consistently enforce the use of standards across all project phases, such as requirements or design.

Table 20: FDD Assessment in Addressing Sub-Attributes of Software Construction KA [3].

Sub-Attribute	Classification (Score)	Aspects of FDD that Satisfy the Attribute
Minimizing Complexity	AS (Adequately Satisfied)	Decomposes large problems into manageable ones.
Response to Changes	AS (Adequately Satisfied)	Supports continuous integration, refactoring, and collective code ownership.
Construction for Verification	AS (Adequately Satisfied)	Utilizes pair programming and unit testing.
Standards in Construction	PS (Partially Satisfied)	Enforces standards primarily during implementation.

Rapid Application Development:
Rapid Application Development (RAD) prioritizes speed through rapid prototyping and iterative development. This approach tackles the Software Construction Knowledge Area (KA) with both strengths and weaknesses.

RAD adequately satisfies Minimizing Complexity. Techniques like iterative prototyping and modular design break down intricate tasks into manageable parts. This iterative process allows for ongoing refinement and simplification of the code, improving understandability and

maintainability. Similarly, RAD adequately satisfies the handling changes sub-attribute due to its iterative and incremental development. Continuous feedback loops and frequent iterations ensure code flexibility and easy modification to meet evolving needs. However, RAD's focus on speed can lead to less emphasis on rigorous testing during construction, resulting in a "Partially Satisfied" rating for constructing for verification, as presented in Table 21. While RAD encourages early testing through prototypes, it might not fully address all verification aspects, potentially introducing undetected bugs and errors.

Finally, RAD's adherence to standards in construction also falls under "Partially Satisfied." The emphasis on speed and flexibility can sometimes lead to deviations from established coding conventions and best practices. Maintaining strict coding standards might require additional effort within the RAD framework.

Table 21: RAD Assessment in Addressing Sub-Attributes of Software Construction KA [3].

Sub-attribute	Classification (Score)	Aspects of RAD that Satisfy the Attribute
Minimizing Complexity	Adequately Satisfied (AS)	Utilizes iterative prototyping and modular design for easier code understanding and management
Response to Changes	Adequately Satisfied (AS)	Emphasizes iterative and incremental development for quick adaptation to changing requirements
Constructing for Verification	Partially Satisfied (PS)	Promotes early testing through prototypes but may overlook comprehensive verification practices
Standards in Construction	Partially Satisfied (PS)	Emphasizes speed and flexibility, potentially leading to deviations from coding standards and best practices

RESULTS AND ANALYSIS

After evaluating agile methodologies across sub-attributes in software requirements and construction, we concisely summarize the assessment results. Table 22 and Figure 1 visually represent the percentage distribution among score classes (Adequately Satisfied, Partially Satisfied) for each methodology, illuminating their strengths and weaknesses effectively.

Table 22: KA Coverage Results of the 5 Agile Methodologies.

SWEBOK KA	XP			Scrum			Kanban			FDD			RAD		
	AS	PS	NS	AS	PS	NS	AS	PS	NS	AS	PS	NS	AS	PS	NS
Software Requirements	60%	40%	0%	40%	60%	0%	60%	40%	0%	80%	20%	0%	40%	60%	0%
Software Construction	50%	50%	0%	50%	25%	25%	50%	25%	25%	75%	25%	0%	50%	50%	0%

Upon analyzing the coverage of each agile method regarding the sub-attributes within the chosen Knowledge Areas (KAs), it becomes evident that XP, Kanban, and FDD have satisfactorily addressed most of the subattributes within the Software Requirements KA. This conclusion is evident in Table 22, where these methods exhibit a notable percentage of "Adequately Satisfied" (AS) classifications across the sub-attributes.

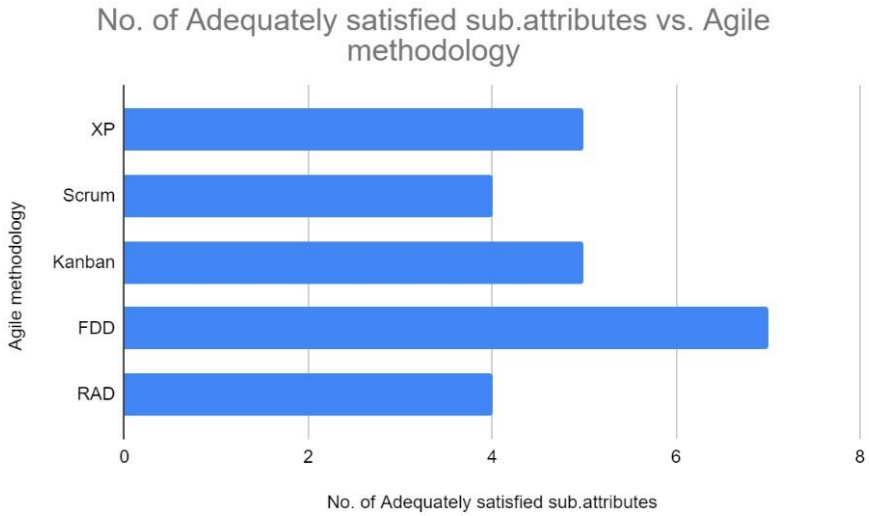

Figure 1: Total Number of Adequately Satisfied Sub-Attributes by Each Agile Methodology.

Examining Table 22 shows that Extreme Programming (XP), Kanban, and Feature Driven Development (FDD) have demonstrated strong performance in the Software Requirements KA. All three methodologies attained "Adequately Satisfied" scores in most sub-attributes. This overall positive performance suggests that these methodologies establish a robust foundation for gathering, analyzing, and validating requirements throughout the software development lifecycle.

XP's emphasis on collaboration, user stories, and iterative development practices such as pair programming and code reviews contributes to its "Adequately Satisfied" scores across most sub-attributes. Similarly, Kanban achieves comparable results with its focus on visual workflow and continuous improvement. Notably, FDD stands out, achieving "Adequately Satisfied" ratings in all sub-attributes for Software Requirements. This noteworthy performance likely stems from FDD's well-defined process for requirement specification, analysis, and validation, coupled with its emphasis on clearly distinguishing between various requirement types.

In contrast, the landscape differs in Software Construction. FDD emerges as the sole methodology to achieve an "Adequately Satisfied" score in 80% of the sub-attributes. This aligns with FDD's emphasis on clear coding practices, unit testing, and code reviews throughout development cycles. Conversely, XP, Kanban, and Scrum methodologies achieved "Adequately Satisfied" scores in only half of the sub-attributes for Software Construction. While these methodologies advocate practices that enhance code quality, the absence of a formally defined coding standard often leads to inconsistencies, potentially affecting long-term maintainability.

CONCLUSIONS AND FUTURE WORK

This study comprehensively evaluated five Agile methodologies (XP, Scrum, Kanban, FDD, and RAD) concerning their suitability for addressing Software Requirements and Software Construction Knowledge Areas (KAs), providing valuable insights for methodology selection based on project characteristics. FDD emerged as the frontrunner, exhibiting high satisfaction

levels (80% and 75%) across both KAs. Its welldefined processes for requirement gathering, analysis, and validation, coupled with an emphasis on clear coding practices and code reviews, position FDD as an optimal choice for projects with complex requirements or a critical need for maintainable code.

XP demonstrated proficiency in Software Requirements (60% satisfaction) with user stories and iterative development practices. However, its lack of a formal coding standard may affect long-term maintainability, resulting in a 50% satisfaction level for Software Construction. XP could suit projects emphasizing early functionality delivery, but additional measures may be necessary to ensure code quality over time.

Scrum and Kanban yielded differing results in Software Requirements, with Scrum achieving a 40% satisfaction level due to its less structured requirement gathering compared to FDD. In contrast, Kanban, lacking a formal requirement process, scored 0%. Nevertheless, both methodologies attained a 50% satisfaction level in Software Construction. Scrum's emphasis on continuous integration practices and Kanban's focus on continuous improvement and the potential use of automated testing contribute to this score. Scrum might suit projects valuing flexibility and adaptability, while Kanban could benefit those emphasizing continuous flow and rapid delivery.

RAD displayed one of the lower overall satisfaction levels. Although achieving a 50% satisfaction level in Software Requirements due to its iterative nature with user involvement, its emphasis on rapid prototyping may negatively impact maintainability, resulting in a 25% satisfaction level in Software Construction. RAD might suit projects with short timeframes and high user involvement, but careful consideration of long-term code quality implications is warranted. While these findings provide valuable insights, it is crucial to acknowledge their context-specific nature.

Future research could further explore agile methodologies by expanding the scope to encompass additional KAs beyond Software Requirements and Software Construction, conducting long-term impact studies to delve deeper into methodology outcomes, and investigating the effectiveness of agile methodology hybrids to address specific project needs.

References

[1]. Al-Saleem, Saleh & Ullah, Hanif. (2015). A Comparative Analysis and Evaluation of Different Agile Software Development Methodologies.

[2]. Fernandes, J. M., & Almeida, M. (2010). Classification and comparison of agile methods. In J. M. Fernandes & M. Almeida (Eds.), Classification and comparison of agile methods [Proceedings of the Seventh International Conference on the Quality of Information and Communications Technology (pp. 391-396)]. IEEE, Porto, Portugal. doi: 10.1109/QUATIC.2010.71: https://doi.org/10.1109/QUATIC.2010.71

[3]. Chowdhury, A. F., & Huda, M. N. (2011). Comparison between adaptive software development and feature-driven development. In Proceedings of 2011 International Conference on Computer Science and Network Technology (pp. 363-367). IEEE. doi: 10.1109/ICCSNT.2011.6181977

[4]. Bourque, R. E. Fairley, and IEEE Computer Society, SWEBOK v.3.0: Guide to the software engineering body of knowledge. IEEE, 2014.

[5]. Beck, K. (2000). Extreme Programming Explained: Embrace Change. Addison-Wesley.

[6]. Azanha, A., Argoud, A. R. T. T., De Camargo, J. B., & Antoniolli, P. D. (2017). Agile project management with Scrum. International Journal of Managing Projects in Business, 10(1), 121–142. https://doi.org/10.1108/ijmpb-06-2016-0054

[7]. Anderson, D. (2010). Kanban: Successful Evolutionary Change for Your Technology Business. Blue Hole Press.

[8]. Arbain, A. F., Ghani, I., & Jeong, S. R. (2014). A Systematic Literature Review on Secure Software Development using Feature Driven Development (FDD) Agile Model. Inteo'nes Jeongbo Haghoe Nonmunji/Inteonet Jeongbo Hakoe Nonmunji, 15(1), 13–27. https://doi.org/10.7472/jksii.2014.15.1.13

[9]. Beynon‐Davies, P., Carne, C., Mackay, H., & Tudhope, D. (1999). Rapid application development (RAD): an empirical review. European Journal of Information Systems, 8(3), 211–223. https://doi.org/10.1057/palgrave.ejis.3000325

[10]. Martin, J. A. (2007). Rapid application development. In Auerbach Publications eBooks. https://doi.org/10.1201/9781420044287-38

[11]. Zasornova, I., Lysenko, S., & Zasornov, O. (2022). Choosing Scrum or Kanban Methodology for Project Management in IT Companies. Computer Systems and Information Technologies, (4), 6–12. https://doi.org/10.31891/csit-2022-4-1

[12]. Brych, V., & Peryt, I. (2019). Basics of business management of domestic economic entities. Scholarly Notes of the «KROK» University, Chapter 6. Management and Marketing, (3)55, 82–93.

[13]. Abrahamsson, P., Salo, O., Ronkainen, J., & Warsta, J. (2002). Agile Software Development Methods: Review and Analysis. https://arxiv.org/abs/1709.08439.

[14]. Parsons, D. (2022). Agile and Lean Software Engineering and the SWEBOK. Paper presented at the 34th International Conference on Software Engineering and Knowledge Engineering (SEKE 2022), Pittsburgh, USA.

[15]. OpenAI. (2025). ChatGPT-4.5 [Large language model] and Grammarly editing software are used to check for grammar & clarity.