Proceedings of the 3rd International Conference on Software Engineering and Machine Learning
DOI: 10.54254/2755-2721/119/2025.21671

Comparative Analysis of Agile Development and Traditional Development

Guojin Tan[1,a,*]

[1]*Harbin University of Science and Technology, Harbin, 150080, China*
a. java.qgj@gmail.com
**corresponding author*

Abstract: As the requirements of engineering projects become more and more stringent, traditional development methods have gradually been unable to meet the requirements of many engineering projects, so agile development has emerged. However, although agile development is very flexible in responding to requirements, it ignores important documents and lacks overall planning., resulting in difficulty in maintenance and lack of stability during development. Therefore, this article focuses on analyzing the advantages and disadvantages of agile development and traditional development, and combines Scrum to tools, so that the developed projects can better meet the needs, be more efficient, be more maintainable and more stable. Both have their own applicable scenarios, and the two can complement each other and hybrid development. This paper finds that agile development has obvious advantages in responding quickly to changes and improving team collaboration, but insufficient documentation and weak planning capabilities may affect later maintenance; traditional development relies on stability and documentation. Advantages, but poor adaptability to changes in requirements limits development efficiency. Based on this, this paper proposes a hybrid development strategy that combines the advantages of the two. Using agile flexibility and traditional standardization complementation can improve development efficiency and stability. In addition, combining agile tools (such as Scrum, and Kanban) with traditional requirements analysis and documentation processes can optimize the management and standardization of the development process. In the end, it provided a new methodology for the software development industry and proposed adaptive research directions for different project sizes and industries.

Keywords: Scrum, Kanban, Agile development, traditional development

1. Introduction

With the rapid development of information technology, software development has become an important means to promote social progress and enhance business competitiveness. However, the diversity and complexity of software development projects make choosing the appropriate development method the focus of the industry. Agile development is favored for its rapid response to changes, while traditional development still occupies an important position due to its structured management method and stability. Although agile development and traditional development have their own advantages, they also have their own limitations. The insufficient documentation and weakening of long-term planning in agile development often lead to maintenance difficulties later in

DOI: 10.54254/2755-2721/119/2025.21671

the project, while traditional development shows low flexibility in the face of frequent requirements changes. This single approach makes it difficult for development teams to comprehensively respond to the needs of complex projects. This paper aims to deeply analyze the advantages and disadvantages of agile development and traditional development, and explore how to reasonably combine the characteristics of the two, give full play to their advantages, and overcome their respective limitations, so as to achieve comprehensive improvement in development efficiency and quality. Through this research, it can not only provide guidance for development teams to select and implement development methods in complex projects, but also provide new ideas and practical frameworks for the improvement of software engineering methodologies, further promoting the continuous progress of the software development industry. This paper first analyzes the characteristics, advantages and disadvantages of agile development and traditional development, then discusses the feasibility and strategies of combining the two, and finally puts forward specific suggestions for optimizing the development process and summarizes the research significance.

2. Overview of Agile Development

2.1. Definition and Origin of Agile Development

Agile development is a people-centered, iterative and step-by-step software development approach that emphasizes flexibility, collaboration and rapid response to change, and is designed to meet customer needs by frequently delivering working software. The core values of agile development include individuals and interactions over processes and tools, available software over complete documents, customer collaboration over contract negotiation, and responsiveness to change over compliance. The background of agile development can be traced back to the 1990s, when the software development industry faced a "software crisis", which was manifested by problems such as project overbudget, overtime, low quality, difficulty in maintenance, and failure to meet customer needs [1]. Although the traditional waterfall model has a clear structure, its linear development model is difficult to cope with rapidly changing needs, resulting in high project failure rates and low customer satisfaction [2]. In order to deal with these problems, the field of software development has begun to explore new methodologies. Lightweight methods such as Extreme Programming (XP), Scrum, and Lean Development (Lean) have gradually emerged, which emphasize iterative development, customer collaboration, and rapid response to changes [3]. In February 2001, 17 software development experts gathered at the Snowbird Ski Resort in Utah, USA, and jointly signed the "Agile Manifesto", formally proposing the concept of agile development [4].

2.2. Core Principles of Agile Development

Agile development follows 12 principles. (1) Early and continuous delivery of valuable software, obtaining timely feedback from customers through frequent interactions with available software, and improving customer satisfaction [5]. (2) Welcoming requirements changes, agile development reduces the cost and risk of requirements changes through flexible iterative processes and simplified change management [6]. (3) Frequent delivery, frequent delivery of working software, delivery cycles ranging from a few weeks to a few months, the shorter the better. Through a delivery strategy of small steps and fast runs, agile development reduces the risk of large-scale integration and increases the frequency of customer feedback [7]. (4) Daily collaboration between business personnel and developers emphasizes communication and close cooperation between cross-functional teams [8]. (5) Build an incentive mechanism with people at the core. Encourage team members to innovate independently by creating an environment of trust and respect [9]. (6) Face-to-face communication encourages direct face-to-face communication between teams to improve information communication efficiency [10]. (7) Available software is the primary measure of progress. Project

DOI: 10.54254/2755-2721/119/2025.21671

progress is measured through actual delivered software functions rather than written documentation [11]. (8) Sustainable development encourages teams to work at a sustainable pace and avoid quality problems caused by excessive overtime [12]. (9) Continue to focus on technical excellence and good design to maintain code quality and maintainability through continuous refactoring and technological innovation [13]. (10) Concise-Minimize unnecessary work, pursue simple design and avoid unnecessary complexity [14]. (11) Self-organizing teams empower teams to make independent decisions and improve innovation capabilities and sense of responsibility [15]. (12) Regular reflection and adjustment Conduct regular team reviews to continuously optimize working methods and processes [16].

2.3. Key Tools for Agile Development

Scrum and Kanban are two popular tools in agile development. With appropriate adjustments, they can effectively make up for the shortcomings of traditional development while retaining the advantages of rigour and maintainability in traditional development. Scrum Agile Development is an agile development framework. It is an incremental, iterative development process that is visible, integrable, and runnable. Different from the traditional waterfall development model, it prefers to perform short and fast version iterations of local modules of a complex system to quickly respond to expected market demand verification. Scrum agile development generally has four inputs/outputs: (1) Analyze, investigate and transform users' needs, and issue product BACKLOG; (2) Product BACKLOG, Sprint planning meeting, Sprint BACKLOG: Split product BACKLOG into Sprint BACKLOG can be refined in this Sprint and managed according to development priorities, updating Sprint BACKLOG status at any time; (3) Sprint BACKLOG, iterative development cycle, deliverable iteration version: Start development work according to the Sprint BACKLOG and update the work task panel to ensure that the overall development progress does not deviate significantly from the preset Sprint burn-down chart. (4) Accept the release version, review meeting, cycle the data report: PM accepts the iterative version of the development delivery based on the product BACKLOG, and releases the iterative version of the product. Collect feedback on Sprint issues, identify root causes, discuss solutions, and improve the Sprint process. Kanban is a visual project management method that originated in Toyota's production system and aims to achieve continuous improvement by limiting the amount of work-in-process (WIP), optimizing work processes, and improving team collaboration efficiency. Kanban tracks the progress of tasks by displaying task status and updating them in real time, such as listing to-do, in-progress, and completed events. WIP limit is one of Kanban's core concepts. It prevents task backlog by limiting the upper limit of the number of tasks processed simultaneously, improves development efficiency, and ensures smooth development. In addition, managing flow optimizes the development process by measuring cycle time and throughput, and sets clear working standards to improve team collaboration efficiency and make the process clear. Optimize the process through daily standing meetings and regular reviews to achieve feedback loops, and adopt incremental optimization strategies to achieve evolutionary change rather than large-scale change.

3. Overview of Traditional Development

3.1. Definition and Basic Process of Traditional Development

Traditional development usually refers to the adoption of linear development methods such as the waterfall model, emphasizing stage division and document driving. The core is to complete the steps of requirements analysis, design, coding, testing and maintenance in order, and enter the next stage after each stage is over. Traditional development generally has a clear process: (1) Requirements analysis: To clarify system functions and user needs, a specific document is usually written to

DOI: 10.54254/2755-2721/119/2025.21671

describe user needs to provide clear plans for subsequent design; (2) System design: Formulate the system architecture and module design, write documents, and design the structural database interfaces required by the system, etc.; (3) Coding implementation: Convert the design into executable code and conduct unit testing to ensure that the code meets the design requirements; (4) Testing: Verify the functions and performance of the system, troubleshoot and fix problems; (5) Deployment: deliver the system to users for use; (6) Maintenance: Ensure stable operation of the system, fix vulnerabilities, and update the system in a timely manner as required.

3.2. Typical Models of Traditional Development

Typical models for traditional development include the waterfall model, the V model, the incremental model, the spiral model, and the iterative model. The basic process described above is a reflection of the original waterfall model, while the V model is an improvement of the waterfall model. It corresponds to the development phase and the testing phase one by one to ensure quality control. The incremental model splits software development into multiple small increments. Each increment is a deliverable software version, and new functions are added for each delivery. Through repeated iteration, a complete system is finally formed. The spiral model combines the structured process of the waterfall model with the iterative idea of the incremental model, and emphasizes risk management. By evaluating technology and managing risks, risks are further reduced. The iterative model emphasizes that software development is an iterative process, with each iteration delivering a partially available software and continuously optimizing it.

4. Comparison and Combination Strategies between Agile Development and Traditional Development

4.1. Advantages, Disadvantages and Differences between Agile Development and Traditional Development

As the name suggests, agile development has the advantage of high flexibility, can quickly respond to changes in requirements, and is highly adaptable. User needs can be adjusted in a timely manner, and products can be delivered quickly through short iterations. During the entire development process, user participation is high. Through frequent delivery, user needs are responded to in a timely manner to ensure that the development results are more relevant to user needs. It also has the advantage of controllable risks. Since each delivery has a product, the project progress can be continuously monitored, thereby reducing the risk of project failure. Through rapid feedback response and close integration of development and testing, problems can be discovered and solved at an early stage. Traditional development has a fixed process. Because it focuses on planning and documentation, it emphasizes detailed requirements analysis, architecture design and planning before development, so that the entire development process has a clear direction, complete documents, and facilitates subsequent maintenance and handover.

Traditional development is highly normative. Because of its standardized development process, it is very suitable when the team wants to complete a large project or a very high-quality scenario. Traditional development is predictable. Through detailed preliminary planning and milestones, the project's time and cost are highly controllable and suitable for a stable demand environment. Most of its development is concentrated in the early stage, which is conducive to solving key problems at one time. Through analyzing and comparing them, we found that agile development often lacks documentation. Due to the emphasis on "working software is better than detailed documentation", necessary design and technical documentation may be lacking in the later stage of development. For long-term maintenance or handover projects, insufficient documentation can become an obstacle. Second, its planning is not as detailed as traditional development.

DOI: 10.54254/2755-2721/119/2025.21671

Agile development is characterized by "adjusting while developing", which may lead to insufficient overall planning and architectural design. You may also focus too much on short-term goals and ignore long-term development. Agile development is highly dependent on the team. Due to the high requirements on the technical and collaborative capabilities of team members, any weak link in the team will significantly affect efficiency and produce a plank effect. It has a low utilization rate of resources. Some members in the project may be idle due to uneven allocation of tasks in a certain iteration, and it is difficult to achieve long-term balance in resource allocation. The disadvantages of traditional development are also obvious. Its adaptability is very poor. Once needs change, detailed planning in the early stage may require large-scale adjustments, and the modification cost is high. The development cycle is long, and the real needs of users may have changed by the time development is completed. Its delivery time will be long, and users will have to wait for a long time to see the results, which will not only make user engagement low, but may also lead to user needs being misunderstood or expectations being met. Because the testing phase is often concentrated in the later stage, its testing lags behind, which may make it difficult to adjust when problems are discovered. Due to its obvious phased nature, some developers will be idle in the non-development phase, resulting in insufficient utilization of their resources.

4.2. Combine the Advantages of Agile Development and Traditional Development

By comparing their different advantages and disadvantages, we found that their different advantages can make up for each other's shortcomings to a certain extent. For example, traditional development can provide detailed documentation for agile development, while agile development can provide flexible delivery and rapid iteration for traditional development. Therefore, we can find strategies to combine their advantages and achieve complementarity.

Strategy 1: Switch development models based on phased tasks. (1) During the project start-up stage (traditional development model).

Detailed requirements analysis, architectural design and documentation. Taking advantage of traditional development, clear goal setting and technical difficulties will be completed in the early stage of the project. The output results include requirements documents, system architecture design documents and technical specifications. (2) Mid-term development phase (agile development model): Use agile development for rapid iteration and functional implementation. Communicate frequently with customers, gradually verify requirements and adjust implementation strategies. The development process iteratively delivers available sub-functional modules to ensure flexibility in project advancement. (3) Project closing stage (traditional development model): Return to the traditional development process and conduct system integration and comprehensive testing. Prepare necessary operation and maintenance documents and technical manuals to provide support for post-maintenance.

In this way, we can use the standardization of traditional development to avoid the problem of unclear requirements in the early stage, use the flexibility of agile development to cope with changes in the medium stage, and ensure stable delivery through traditional development in the later stage.

Strategy 2: Hybrid development methods

Goal: Use both agile development and traditional development in the same project, and adopt different strategies for different modules.

(1) Core module: Use traditional development models to carry out detailed planning and documentation to ensure high quality and high stability.

(2) Non-core modules: Use an agile development model to quickly deliver the functions required by users.

DOI: 10.54254/2755-2721/119/2025.21671

(3) Collaboration mechanism: Use tools (such as Jira or Confluence) to seamlessly connect the work of agile teams with traditional teams, ensuring the flow of information between the two development methods.

Strategy 3: (1) Introduce a document management method that combines agile and traditional methods to add lightweight documents to agile development: Add simple design instructions to each user story. Use templated documentation to record technical decisions and design ideas. Simplify some documents in traditional development: For functional modules with low stability requirements, use the lightweight document method of agile development. Use tools such as Confluence to uniformly manage documents:

Store dynamic documents for agile development and formal documents for traditional development in a unified knowledge base.

Strategy 4: Introducing Scrum Kanban.

Scrum can flexibly adjust requirements in the middle and late stages of a project, while ensuring that iterative deliverables meet customer needs. By appropriately introducing traditional development methods, Scrum can solve its problems of insufficient documentation and insufficient long-term planning.

Application strategy: (1) Detailed documentation combined with traditional development: Before the start of Sprint, requirements documents (PRD) and technical design documents common in traditional development can serve as a reference basis. After each iteration is completed, supplement brief documents to record the functional and technical details of the implementation to make up for the lack of documentation in agile development. (2) Appropriately introduce a milestone checking mechanism for traditional development: Introduce a phased review of traditional development after several Sprints to assess whether the overall progress of the project is in line with long-term goals. (3) Plan for stable core project functions: Design key parts of the project (such as core modules or high-risk parts) in advance in the traditional development phase, and then complete the low-risk parts through Scrum iteration. (4) Division of labor between agile and traditional personnel: Agile teams focus on functional realization, while traditional teams focus on architectural design and technical difficulties.

Kanban emphasizes real-time management of tasks, which is suitable for task flow management and delivery control in traditional development, and can make up for the problem of opaque task schedules in traditional development.

Application strategy: Use Kanban in the requirements analysis stage of traditional development: decompose requirements analysis tasks into detailed work items and visualize progress through Kanban. Ensure that requirements analysis is completed step by step according to priority to avoid waste of resources caused by large-scale requirements analysis. (1) Integrate traditional quality control processes in agile development: Add quality inspection links (such as code review, and integration testing) to Kanban as a necessary step of the task. Embed traditional development testing processes into Kanban's workflow to ensure delivery quality. (2) Sprint combined with Scrum: Use Kanban to manage task status in detail during the iteration cycle of Scrum to avoid too extensive task allocation in Scrum. (3) Cross-team collaboration: Use Kanban to manage collaboration between development and test teams to ensure clear and transparent task switching.

5. Conclusion

Through an analysis of the advantages and disadvantages of agile development and traditional development, we can find that agile development has significant advantages in responding quickly to changes, improving user satisfaction, and enhancing team collaboration. However, its shortcomings, such as insufficient documentation and weak long-term planning capabilities, may have a negative impact on the project. Maintenance has a negative impact. Traditional development has stability and

DOI: 10.54254/2755-2721/119/2025.21671

comprehensive documentation as its core advantages, but its low adaptability to changes in requirements and long feedback cycles limit development efficiency. On this basis, this paper proposes a development strategy that combines the advantages of the two, adopting agile flexibility and traditional standardization complementation, which can effectively improve development efficiency and stability.

The combination of agile development and traditional development is not a simple technical choice, but a management innovation that conforms to the development trend of modern software engineering. By optimizing the combined application of the two, the development team's adaptability in complex environments can be significantly improved to meet changing customer needs. At the same time, this research provides a new methodological reference for the software development industry and helps promote the further improvement of industry standards and practices.

In practice, flexible management of the development process can be achieved by introducing agile tools such as Scrum and Kanban, and at the same time, the standardization and traceability of development can be strengthened with the help of traditional development requirements analysis and documented processes. Specifically, the small-batch iteration method suitable for agile development can run through the entire process of project development, while the comprehensive requirements analysis and phased quality assessment of traditional development should be given full attention during the project launch and delivery stages.

Although this paper makes a preliminary discussion on the combination of agile development and traditional development, there are still many issues worthy of further study. For example, how to design suitable hybrid development models for projects of different sizes and industries? How to further improve the efficiency of combining the two methods through automated tools? Solving these problems will provide more possibilities for innovation and optimization of software development methods.

References

[1] *Beck K., Beedle M., van Bennekum A., Cockburn A., Cunningham, W., Fowler, M., ... & Thomas, D. (2001). Manifesto for Agile Software Development. Retrieved from http://agilemanifesto.org/*

[2] *Boehm B. W. (1991). Software Engineering Economics. Prentice-Hall, pp.1-150.*

[3] *Royce W. W. (1970). Managing the development of large software systems. Proceedings of IEEE WESCON, pp.1-9.*

[4] *Schwaber K., & Beedle M. (2002). Agile Software Development with Scrum. Prentice-Hall, pp.1-328.*

[5] *Cockburn A., Highsmith J. (2001). Agile software development: The people factor. Computer, 34(11), 131-133.*

[6] *Fowler M., Highsmith J. (2001). The agile manifesto. Software Development, 9(8), 28-35.*

[7] *Schwaber K., Beedle, M. (2002). Agile Software Development with Scrum. Prentice Hall, pp.1-158.*

[8] *Cockburn A. (2002). Agile Software Development. Addison-Wesley, pp.1-304.*

[9] *Dingsøyr T., Nerur S., Balijepally V., & Moe N. B. (2012). A decade of agile methodologies: Towards explaining agile software development. Journal of Systems and Software, 85(6), 1213-1221.*

[10] *Beck K., Beedle, M., van Bennekum A., Cockburn A., Cunningham, W., Fowler, M., ... & Thomas, D. (2001). Manifesto for Agile Software Development. Retrieved from http://agilemanifesto.org.*

[11] *Schwaber K. (2004). Agile Project Management with Scrum. Microsoft Press, pp.1-192.*

[12] *Poppendieck M., Poppendieck T. (2003). Lean Software Development: An Agile Toolkit. Addison-Wesley, pp.1-192.*

[13] *Beck K. (2000). Extreme Programming Explained: Embrace Change. Addison-Wesley, pp.1-189.*

[14] *Martin, R. C. (2008). Clean Code: A Handbook of Agile Software Craftsmanship. Prentice Hall, pp.1-464.*

[15] *Takeuchi H., & Nonaka I. (1986). The new new product development game. Harvard Business Review, 64(1), 137-146.*

[16] *Schwaber K., & Sutherland J. (2011). The Scrum Guide. Scrum Alliance, pp.1-17.*