Proceedings of CONF-SPML 2026 Symposium: The 2nd Neural Computing and Applications Workshop 2025
DOI: 10.54254/2755-2721/2026.TJ27796

Comparative Analysis of Software Development Methodologies

Hanyang Cao

College of Computer Science and Technology, Zhejiang University, Hangzhou, China
3220104064@zju.edu.cn

Abstract. The contemporary software development landscape is characterized by a proliferation of methodologies, yet academic discourse predominantly centers on the examination of individual models in isolation, rather than undertaking holistic comparative analyses. This gap highlights the need for a structured evaluation of different approaches to guide practitioners in selecting optimal models for diverse project requirements. This study systematically categorizes and compares various software development models—including flow-based, structured, iteration-based, object-oriented, and composite models—to assess their flexibility, risk management, expertise requirements, and applicability across project sizes and environments. Employing a literature review approach, the research analyzes existing models (e.g., Waterfall, Agile, DevOps) and evaluates them across eight critical dimensions: flexibility, risk, time, expertise, project size, customer involvement, delivery frequency, and quality assurance mechanisms. The findings reveal that agile models (e.g., Scrum, XP) excel in flexibility, customer engagement, and iterative delivery, making them ideal for dynamic projects. Traditional models (e.g., Waterfall) suit stable, small-scale projects but lack adaptability. High-risk projects benefit from Spiral and MDDF, while DevOps and Crystal methodologies balance structure and flexibility. The study underscores the growing trend toward flexible, collaborative approaches in modern software development, emphasizing the importance of context-specific model selection to enhance efficiency and outcomes.

Keywords: Software Development, Development Methodology, Project Management, Comparison, Models

1. Introduction

In recent years, the software development landscape has witnessed the emergence of numerous methodologies and models. However, existing literature predominantly focuses on in-depth analysis of individual models, lacking comprehensive comparative studies across the entire spectrum of development approaches. While Chandra, V. [1] examined 21 existing models and Saeed, S., Jhanjhi, N. Z., Naqvi, M., & Humayun, M. [2] analyzed 20 prominent models—including seminal approaches such as Waterfall, Spiral, and V-Model—these studies primarily present catalog-style reviews without providing systematic classifications. Notably, Despa, M. L. [3] pioneered a categorization framework by organizing seven widely adopted models into four distinct types: flow-based, structured, iteration-based, and object-oriented approaches, which will serve as the taxonomic foundation for this research. Based on these four categorizations, this article will integrate and compare existing models, analyzing the strengths, weaknesses, and application scenarios of each model. It will first provide a brief introduction to software, then analyze the characteristics of each model, and finally compare their application scenarios. Employing a systematic literature review methodology, this research aims to contribute to the theoretical advancement of software development methodologies while providing practical guidance for practitioners to optimize development processes and enhance project outcomes.

2. Software development stage

Software development, also known as the Software Development Life Cycle (SDLC), is divided into stages to ensure an efficient, orderly process that delivers reliable software meeting user needs. Software architecture and development stage encompasses the set of significant decisions about the organization of a software system [4]. Despite methodological variations, six fundamental stages are universally recognized in contemporary software engineering practice:

Requirements Analysis: This critical initial phase involves gathering and clarifying user needs and defining functional and performance requirements. Tasks include user interviews, requirements categorization, documentation, and verification to ensure clarity, feasibility, and alignment with project goals. It minimizes risks, improves communication, and lays a foundation for design, development, and testing.

System Design: This phase plans the software architecture, defining system structure, module breakdown, and data flow. Core activities encompass functional specification of modules, interface protocol design, technology stack selection, and creation of detailed design documentation. It guides implementation, reduces coding errors, optimizes resources, and influences organization control and quality outcomes. This phase provides critical guidance for implementation activities, reduces coding anomalies, optimizes resource allocation, and significantly influences organizational control mechanisms and quality assurance outcomes [5]. Notably, user interaction during this design phase substantially impacts design quality, organizational control strategies, and subsequent system implementation success [5].

Implementation: Based on design documents, this phase translates plans into actual code using programming languages. Code must be well-structured, maintainable, documented, and unit-tested to verify module correctness. This core step transforms abstract designs into tangible software, ensuring functionality and setting the stage for testing and maintenance.

Testing: Testing validates functionality, detects defects, and assesses performance, compatibility, security, user experience, and regression. Systematic testing ensures the software meets requirements, reduces post-release defects, lowers maintenance costs, mitigates risks, and builds user trust for continuous improvement.

Deployment: This stage releases the software to a production environment, including installation, configuration, and rollout support, transitioning it from development to actual use. Deployment ensures stable, secure operation, fulfilling user needs and enabling further maintenance and optimization.

Maintenance: Post-deployment, maintenance involves bug fixes, updates, and enhancements to keep software effective, secure, and aligned with evolving business or technical contexts. This extends software life, improves user experience, and maximizes value.

Collectively, these phases constitute a systematic framework that guides software systems from conceptualization through deployment and sustained operational support.

3. Models theories

Software development life cycle (SDLC) models exhibit considerable variation based on specific development environments and application contexts, making the selection of an appropriate model critically important [6]. They can be categorized into flow-based, structured, iteration-based, object-oriented, and composite models.

Flow-based Models like waterfall model, as is shown in Fig 1, operate through sequential, linear progression encompassing distinct phases including requirements analysis, system design, implementation, verification testing, and deployment. Methodologically, these approaches emphasize unidirectional execution, stringent process governance, and comprehensive documentation, rendering them particularly suitable for projects characterized by well-defined and stable requirements. Waterfall model is the classical example, simple but inflexible, requiring well-defined requirements early and limiting iteration and feedback [7]. V-Model links each development phase with a corresponding testing phase, ensuring thorough verification but lacking flexibility for rapid changes. Cleanroom model focuses on high quality and reliability through formal verification and rigorous processes, ideal for critical systems. The Microsoft Solutions Framework balances standardized processes with practical flexibility and risk management, suitable for medium to large projects with evolving requirements.

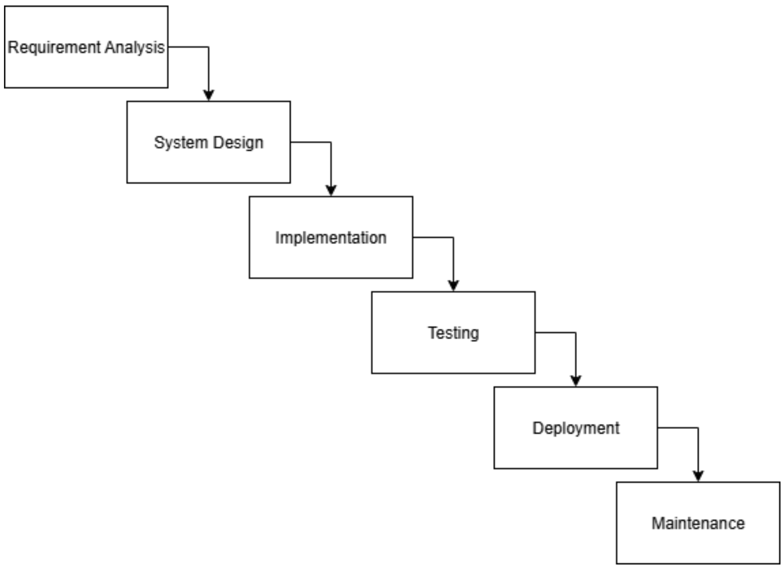

Figure 1. Flow-based Models

Structured Models emphasize a systematic, phased approach with clear requirements and rigorous design. These methodologies prioritize standardization and process control, making them particularly suitable for projects with well-specified requirements and minimal anticipated changes [8]. Joint Application Design (JAD) enhances user participation and communication through centralized meetings, improving efficiency and specification quality. The Dynamic Systems Development Methodology (DSDM) effectively integrates structured approaches with agile principles, supporting rapid development cycles and continuous stakeholder involvement while maintaining an appropriate balance between control and flexibility.

Iterative Models decompose the development process into multiple short cycles, each encompassing planning, design, implementation, and verification phases, thereby delivering incremental functional capabilities. As Fig 2 shows, they offer flexibility to adjust to feedback and encourage collaboration. Prototyping quickly develops preliminary versions to validate ideas and gather user feedback, useful early in design but with risks of overemphasis on visuals. Iterative and Incremental model delivers working software progressively with continuous improvement but demands skilled management and staff. Spiral model combines risk assessment with iterative development, ideal for complex, high-risk projects but costly and complex. Rapid Application Development (RAD) focuses on fast prototyping and iterative delivery for rapid, efficient development of small to medium projects. Adaptive Software Development (ASD) emphasizes organizational adaptability, self-organizing teams, and continuous learning processes within dynamic environments, presenting implementation challenges for teams with limited experience. Lastly, DevOps fosters collaboration between development and operations, enabling continuous integration, delivery, and automated processes for fast, high-quality releases.

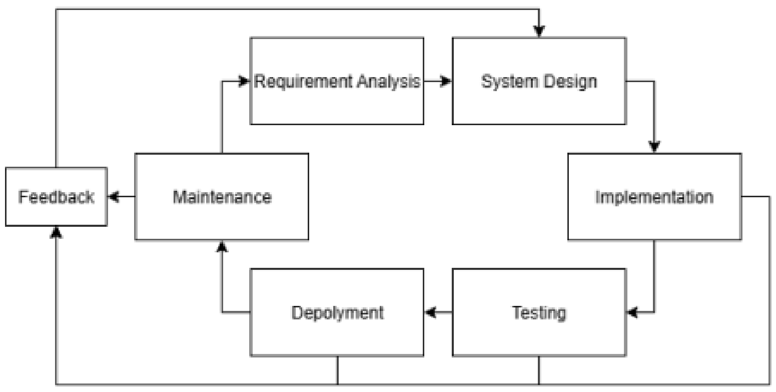

Figure 2. Iteration-based Models

Object-Oriented Models emphasize modularity, reusability, and maintainability through core principles including encapsulation, inheritance, and polymorphism. Model-Driven Development (MDD) concentrates on creating abstract models that automate software design and implementation processes, thereby enhancing development efficiency and product quality. While MDD demonstrates particular effectiveness for complex, multi-platform business systems, its implementation depends heavily on specialized tooling and may exhibit limited flexibility. Crystal methodology represents a lightweight agile framework centered on human factors, offering tailored variations based on project scale and complexity parameters. It values strong team communication, iterative development, frequent delivery of working software, and adaptability, suitable for small to medium projects with high technical and communication demands.

Composite Models combine multiple development approaches to adapt to complex, changing requirements, balancing standardization with flexibility. The Rational Unified Process (RUP) functions as an iterative, phased framework specifically designed for large-scale complex projects requiring robust architectural foundations and experienced development teams. Lean Software Development applies lean principles to maximize value by eliminating waste, fitting dynamic, innovation-driven environments but needs strong cultural commitment [9]. Agile methodology represents a comprehensive software development approach emphasizing flexibility, collaborative processes, and continuous improvement mechanisms as shown in Fig 3. Its core philosophy is to rapidly respond to change and deliver high-quality software through iterative and incremental development. While primarily characterized by iterative approaches, agile methodologies also incorporate object-oriented design principles and collaborative team structures.

Open Source Development operates as a collaborative, publicly accessible model promoting technological innovation and broad knowledge sharing, serving as the foundation for critical systems including Linux and Apache.

Agile methodology encompasses numerous variations and refinements, with several prominent models representing distinct approaches to software development. Extreme Programming (XP) focuses on frequent releases, pair programming, and test-driven development for rapid quality delivery by skilled teams but suits less for large/distributed groups. Scrum promotes teamwork, self-organization, and short iterations, ideal for small to medium teams in fast-changing fields but demands discipline and clear requirements. Test-Driven Development (TDD) writes tests before code for early defect detection, requiring a strong testing culture but slowing initial progress. Behavior-Driven Development (BDD) extends TDD with behavior-focused, testable specifications to improve collaboration and automate acceptance testing. Feature-Driven Development (FDD) organizes work by features for projects with stable requirements but lacks flexibility for changes.

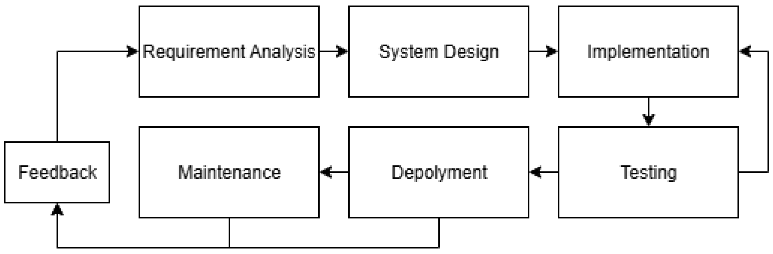

Figure 3. Agile methodology

4. Comparison

The subsequent analysis will systematically evaluate these methodologies across eight critical dimensions, employing a multi-level assessment framework for comprehensive comparative analysis:

. Flexibility(Flex): No(N), Few(F), High(H)

. Risk: High(H), Moderate(M), Low(L)

. Time Required(T Req): Long(L), Moderate(M), Short(S), Variable(V)

. Expertise Required(Exp Req): Few(F), Moderate(M), High(H)

. Suitable Project Size(Pro Size): Small(S), Medium(M), Large(L)

. Customer Involvement(Cus Inv): Low(L), Medium(M), High(H), Very High(VH)

. Delivery Frequency(Del Freq): Low(L), Medium(M), High(H), Very High(VH), Variable(V), Continuous(C)

. Quality Assurance Mechanisms(QA Mech)

Table 1. Model comparison

Model	Flex	Risk	T Req	Exp Req	Pro Size	Cus Inv	Del Freq	QA Mech
Waterfall	N	H	L	M	S	L	L	Formal, Heavy
V-Model	N	H	L	M	S	L	H	Extensive Testing
Cleanroom	F	H	L	H	M	M	L	Strong Formal QA
MSF	F	M	M	H	L	M	M	Built-in Testing
JAD	F	L	S	M	S	M	VH	Moderate
DSDM	F	L	M	M	S, M	H	M	Integrated Testing
Prototyping	H	M	S	M	S	M	L	Informal Testing
I & I	H	M	M	M	M, L	M	M	Moderate
Spiral	H	L	L	H	L	M	H	Strong QA
RAD	H	M	S	M	S, M	L	H	Moderate
ASD	H	M	M	H	M, L	M	H	High
DevOps	H	M	V	H	M, L	M	C	Integrated QA
MDDF	F	H	L	H	M, L	M	M	Moderate
Crystal	H	M	M	M	S, M	M	H	Moderate
RUP	F	M	M	H	M	M	M	Embedded QA
LSD	H	M	M	H	M, L	H	M	Moderate
XP	H	M	S	H	S, M	H	VH	Continuous Testing
Scrum	H	M	S	M	S, M, L	H	VH	Continuous QA
TDD	H	M	M	H	S, M, L	M	L	Continuous Testing
BDD	H	M	M	H	S, M, L	H	M	Continuous testing
FDD	H	M	M	H	M, L	M	M	Moderate
OSSD	H	L	V	M	S, M, L	M	V	Moderate

From Table 1, among software development models, agile models (such as XP, Scrum, TDD, BDD, and FDD) offer high flexibility, support frequent delivery and continuous testing, and high customer engagement. These methodologies exhibit adaptability across project scales, from small to large implementations, while particularly excelling in development contexts demanding rapid iteration and high responsiveness to changing requirements.

Traditional models such as Waterfall and the V-model offer less flexibility and are suitable for small to medium-sized projects as Table 1 shows. They employ linear phases and rigorous quality assurance, but have limited risk mitigation capabilities and longer delivery cycles.

The Spiral model and MDDF effectively manage high-risk projects through iterative evaluation. Cleanroom, MDDF, and DevOps require higher levels of technical expertise, while JAD and prototyping require less expertise.

In conclusion, each software development methodology possesses distinct advantages, and selection should be based on careful consideration of project-specific factors including scale, risk profile, customer engagement requirements, and delivery velocity.

5. Conclusion

Software development models vary significantly in terms of flexibility. Traditional waterfall and V-model methodologies are classified as inflexible approaches, characterized by their rigid processes and limited capacity for adaptation. Cleanroom methods, RUP, and the Microsoft Solution Framework, on the other hand, offer a limited degree of flexibility, possessing a certain degree of adaptability while remaining relatively structured. Highly flexible models such as prototyping, iterative incremental methods, spiral models, and agile-related methods (such as Scrum, XP, and DevOps) exhibit rapid adaptability to change and emphasize continuous feedback and adjustment. Overall, software development methods are trending toward greater flexibility. Particularly within modern, complex, and dynamically evolving project environments, highly adaptable methodologies like Agile and DevOps demonstrate enhanced capability to accommodate changing requirements, thereby improving development efficiency and software quality outcomes. This demonstrates the increasing importance of flexibility in software development, and models that accommodate flexibility are more aligned with current trends in the software industry. It is anticipated that future research and practice will continue to yield innovative software development methodologies and models, further enhancing the efficiency and effectiveness of software development processes.

References

[1] Chandra, V. (2015). Comparison between various software development methodologies. International Journal of Computer Applications, 131(9), 7-10.
[2] Saeed, S., Jhanjhi, N. Z., Naqvi, M., & Humayun, M. (2019). Analysis of software development methodologies. International Journal of Computing and Digital Systems, 8(5), 446-460.
[3] Despa, M. L. (2014). Comparative study on software development methodologies. Database systems journal, 5(3).
[4] Shaw, M., & Garlan, D. (1996). Perspectives on an emerging discipline. Perspectives on an Emerging Discipline, Prentice-Hall.
[5] Boland Jr, R. J. (1978). The process and product of system design. Management science, 24(9), 887-898.
[6] Ruparelia, N. B. (2010). Software development lifecycle models. ACM SIGSOFT Software Engineering Notes, 35(3), 8-13.
[7] Model, W. (2015). Waterfall model. Luettavissa: http: //www. waterfall-model. com/. Luettu, 3.
[8] Geoffrion, A. M. (1987). An introduction to structured modeling. Management Science, 33(5), 547-588.
[9] Hofmann, A. (2013). LSD: My problem child. Oxford University Press, USA.