AIDF: A Proposed Adaptive Intent-Driven Framework for Software Development in the AI Era

Dr Ashiq VM[1], Mufeeda K[2]

[1]Assistant Professor Department of Computer Science Malabar College of Advanced Studies Vengara

[2]Assistant Professor Department of Computer Science Jamiya Nadwiyya Arts and Science College Edavanna

DOI: https://doi.org/10.47772/IJRISS.2026.102000007

Received: 14 October 2026; Accepted: 19 October 2026; Published: 19 November 2026

ABSTRACT

Software development has undergone substantial transformation from sequential lifecycle models such as the Waterfall model to iterative and adaptive approaches including Agile, Spiral, DevOps and continuous delivery. However, the emergence of generative artificial intelligence, AI-assisted programming, automated testing, cloud-native infrastructure and continuous feedback is changing the nature of software development itself. Developers are increasingly able to translate natural-language intentions into software components, generate and modify code, create tests, analyze defects and obtain development recommendations with limited manual intervention. Consequently, software development can no longer be adequately represented as a fixed sequence of predefined phases.

This paper proposes the Adaptive Intent-Driven Framework (AIDF), a conceptual software development methodology designed for the emerging AI-assisted development environment. Unlike traditional lifecycle models that primarily organize development activities around phases, AIDF organizes the lifecycle around continuously evolving user intent, automated assistance, human validation, risk assessment, software quality and real-world feedback. The framework introduces six interconnected dimensions: Intent Definition, Intelligent Planning, Adaptive Construction, Continuous Validation, Controlled Release, and Feedback-Driven Evolution. Artificial intelligence functions as an active development collaborator rather than merely as an isolated coding tool, while human developers retain responsibility for requirements interpretation, architectural decisions, validation, security, ethics and final acceptance.

The paper compares AIDF with Waterfall, Spiral, Agile, DevOps and contemporary AI-driven development approaches. It argues that the principal contribution of AIDF is not the replacement of existing engineering practices but their reorganization around intent, adaptability and continuous intelligence. A research agenda is proposed to empirically evaluate AIDF using development velocity, defect density, change adaptability, testing effectiveness, developer effort, security incidents and user satisfaction. The framework provides a conceptual foundation for investigating how software engineering methodologies may evolve from phase-oriented development toward continuously adaptive human-AI software engineering.

Keywords: Software Development Methodology, Software Development Life Cycle, Artificial Intelligence, Generative AI, Adaptive Development, Human-AI Collaboration, AI-Assisted Programming, Software Engineering, Continuous Feedback

INTRODUCTION

Software engineering has traditionally relied on structured development methodologies to control complexity, coordinate teams and improve software quality. The Waterfall model introduced a sequential development structure in which requirements, design, implementation, testing and maintenance were treated as relatively distinct phases. Later approaches, including the Spiral model, emphasized iterative

development and risk management. Agile methodologies subsequently promoted short development cycles, customer collaboration and responsiveness to changing requirements.

DevOps extended this evolution by connecting development and operations through automation, continuous integration, continuous delivery and operational feedback. These developments demonstrate that software engineering methodologies have historically evolved in response to changes in technology, project complexity and organizational requirements.

The current development environment introduces another significant transition.

Generative artificial intelligence can now support requirements interpretation, code generation, code completion, refactoring, debugging, test generation, documentation and software analysis. Recent empirical research demonstrates that generative AI is becoming increasingly integrated into software engineering activities. At the same time, research has identified concerns regarding verification, code quality, security, maintainability and the management of rapidly generated software changes.

The 2025 DORA research on AI-assisted software development describes AI as an amplifier of organizational capabilities and weaknesses. Its findings indicate that simply introducing AI does not automatically produce a successful software delivery system; organizational practices, platforms, testing, version control and development processes remain important.

This creates an important methodological question:

If software can now be generated, evaluated and modified continuously with assistance from intelligent systems, should software development continue to be represented primarily through fixed lifecycle phases?

This paper proposes that a new conceptual perspective is required.

The proposed Adaptive Intent-Driven Framework (AIDF) does not argue that Waterfall, Spiral, Agile or DevOps are obsolete. Instead, it argues that these methodologies were designed around particular assumptions about how development work is performed. AI-assisted development changes several of those assumptions.

In particular, the distinction between requirements, implementation and testing is becoming less rigid. A developer can provide an intention, obtain a generated implementation, evaluate it, modify the intention, regenerate components and validate the result within a single development cycle.

Therefore, this paper proposes moving from a **phase-oriented lifecycle** toward an **intent-oriented adaptive lifecycle**.

Evolution of Software Development Methodologies

Waterfall

The Waterfall model represents software development as a sequence of relatively predefined stages. Requirements are established before design and implementation, followed by testing and maintenance.

Its major advantage is process clarity. Documentation, milestones and responsibilities can be established in advance.

However, the model becomes difficult to manage when requirements change substantially after implementation begins. Returning to earlier stages may involve significant cost and coordination.

Spiral Model

The Spiral model introduced a stronger emphasis on iterative development and risk analysis. Each cycle includes planning, risk assessment, engineering and evaluation.

The model is particularly useful for complex and high-risk projects because risk analysis becomes an explicit part of the lifecycle.

Nevertheless, the process remains organized around recognizable development cycles. The emerging AI-assisted environment can potentially make these cycles much shorter and more fluid.

Agile Development

Agile methodologies changed software development by emphasizing iterative delivery, customer collaboration and responsiveness to change.

Instead of attempting to define the entire product before development, teams progressively refine requirements and deliver working software.

Agile therefore represents an important movement toward adaptability.

However, many contemporary AI-assisted environments introduce an additional level of adaptability in which software components, tests and documentation can be generated or modified dynamically during development.

DevOps and Continuous Delivery

DevOps connects development with operations and encourages automation, continuous integration, continuous delivery and rapid feedback.

This represents another major shift from a static development process toward a continuous software delivery system.

Modern software engineering therefore already contains many elements required for an adaptive methodology.

AIDF builds on these developments but changes the organizing principle from **delivery pipeline** to **evolving intent**.

The Emerging AI Era of Software Development

Generative AI is increasingly being used across the software development lifecycle.

AI systems can assist with:

- requirements interpretation;
- software architecture suggestions;
- code generation;
- code completion;
- refactoring;
- debugging;

- test generation;
- documentation;
- vulnerability identification;
- database queries;
- API development;
- code explanation; and
- maintenance activities.

Recent DORA research reports widespread AI use among technology professionals and emphasizes that AI's effects depend heavily on the underlying software delivery system.

Research also indicates an important tension: AI can reduce the effort required to generate code, but organizations must strengthen review, testing and validation because rapid generation can increase the volume and speed of changes.

Therefore, the methodological challenge is not simply:

How can AI write more code?

It is:

How should the entire development process be reorganized when AI can participate in multiple development activities?

AIDF addresses this question.

The Adaptive Intent-Driven Framework

Concept

The Adaptive Intent-Driven Framework defines software development as a continuously evolving interaction between:

1. Human intent;
2. AI-supported interpretation;
3. adaptive planning;
4. software construction;
5. automated and human validation;
6. controlled deployment; and
7. real-world feedback.

The framework can be represented as:

Intent → Intelligence → Plan → Build → Validate → Release → Feedback → Updated Intent

The final stage does not terminate the lifecycle. Instead, feedback modifies the next development intention.

Consequently, the lifecycle is represented as a continuously evolving loop rather than a linear sequence.

Components of AIDF

Stage 1: Intent Definition

The lifecycle begins with the desired outcome rather than a complete technical specification.

The intent may be expressed through:

- natural-language requirements;
- business objectives;
- user problems;
- functional goals;
- quality expectations;
- constraints;
- security requirements; and
- organizational priorities.

For example:

"Develop a student attendance system that allows teachers to record attendance using mobile devices and provides administrators with real-time reports."

The initial intention does not need to specify every implementation detail.

Stage 2: Intelligent Interpretation

AI-assisted tools analyze the stated intention and identify:

- functional requirements;
- non-functional requirements;
- possible ambiguities;
- dependencies;
- security concerns;
- data requirements;
- possible architecture options; and
- questions requiring human clarification.

The important principle is that AI does not independently establish the final requirements.

Human validation remains mandatory.

5.3 Stage 3: Adaptive Planning

Once the intent is interpreted, the system creates an adaptive development plan.

The plan may include:

- architecture;
- modules;
- interfaces;
- database structures;
- APIs;
- testing requirements;
- security controls;
- deployment requirements; and
- measurable acceptance criteria.

Unlike traditional planning, the plan is treated as a living artifact.

When requirements change, the plan can be regenerated or modified rather than being considered permanently fixed.

Stage 4: Adaptive Construction

Development occurs through human-AI collaboration.

AI may assist with:

- generating initial code;
- producing test cases;
- creating documentation;
- identifying implementation alternatives;
- refactoring;
- debugging; and
- generating configuration files.

Developers review, modify and integrate the generated outputs.

AIDF therefore does not propose replacing software developers with AI.

Instead, it proposes changing the developer's role from primarily writing every implementation artifact manually toward **orchestrating, evaluating, validating and governing intelligent development tools**.

Stage 5: Continuous Validation

Every generated or modified component should pass appropriate validation.

Validation can include:

- unit testing;
- integration testing;
- security testing;
- static analysis;
- performance testing;
- code review;
- AI-assisted verification;
- human review; and
- requirements traceability.

A critical principle of AIDF is:

Generation without validation is incomplete development.

This is particularly important because AI-generated code may appear plausible while containing functional, architectural or security problems.

Stage 6: Controlled Release

Validated changes are released using appropriate deployment mechanisms.

Depending on the system, these may include:

- continuous integration;
- continuous delivery;
- staged deployment;
- feature flags;
- canary releases;
- automated rollback; and
- production monitoring.

Release decisions remain subject to quality and risk thresholds.

Stage 7: Real-World Feedback

After deployment, information is collected from:

- users;
- application logs;
- performance monitoring;
- security monitoring;
- error reports;
- business metrics;
- support requests; and
- operational data.

This information becomes part of the next development cycle.

Therefore:

Deployment is not the end of development.

It is a source of new information about whether the software actually satisfies its intended purpose.

The Core Principle: Intent Rather Than Phase

The most important conceptual difference between AIDF and conventional lifecycle models is the unit around which the process is organized.

Traditional approaches primarily ask:

"Which development phase are we currently in?"

AIDF asks:

"What user or business intent are we currently trying to satisfy, and what evidence shows that we are satisfying it?"

This distinction becomes particularly important when AI can simultaneously support requirements analysis, coding, testing and documentation.

The boundaries between phases become increasingly fluid.

AIDF Compared with Existing Methodologies

Characteristic	Waterfall	Spiral	Agile	DevOps	AIDF
Primary organizing principle	Sequential phases	Risk-driven cycles	Iterations	Continuous delivery	Evolving intent
Requirement change	Relatively expensive	Managed through cycles	Expected	Continuous	Continuously incorporated

AI role	Optional	Optional	Supporting	Supporting/automated	Integrated collaborator
Development cycle	Linear	Iterative	Iterative	Continuous	Adaptive loop
Feedback	Mostly later stages	Each cycle	Frequent	Continuous	Continuous + intent evolution
Automation	Limited	Moderate	Moderate	High	AI + automation
Human role	Execute defined process	Manage risks	Collaborate and prioritize	Develop and operate	Direct, validate and govern AI-assisted development
Testing	Dedicated phase	Each cycle	Continuous within iterations	Automated/continuous	Continuous multi-layer validation
Planning	Mostly upfront	Repeated	Incremental	Continuous	Dynamic and adaptive

This comparison demonstrates that AIDF is not simply another name for Agile or DevOps.

Agile emphasizes adaptability in product development.

DevOps emphasizes integration between development and operations.

AIDF proposes **intent-centered adaptation supported by AI across the development lifecycle**.

Human-AI Collaboration Model

AIDF requires a clear division of responsibilities.

AI responsibilities

AI may:

- generate candidate implementations;
- analyze requirements;
- propose architectures;
- generate tests;
- identify defects;
- explain code;
- recommend improvements; and
- assist documentation.

Human responsibilities

Humans remain responsible for:

- defining organizational goals;
- approving requirements;

- architectural accountability;
- ethical decisions;
- security decisions;
- quality acceptance;
- regulatory compliance;
- risk management; and
- final deployment authorization.

This principle can be expressed as:

AI proposes → Human evaluates → System validates → Organization decides.

This prevents AIDF from becoming an uncontrolled autonomous coding model.

Why Existing Methodologies Alone May Be Insufficient

It would be inaccurate to claim that traditional methodologies have become useless.

Waterfall remains appropriate for some projects with highly stable and regulated requirements.

Spiral remains useful for risk-intensive projects.

Agile remains highly relevant for iterative product development.

DevOps remains essential for continuous software delivery.

The argument of this paper is different.

The emergence of AI introduces a new development capability that cuts across these methodologies.

For example, an AI system may simultaneously:

- interpret a requirement;
- propose an architecture;
- generate code;
- generate tests;
- analyze vulnerabilities;
- produce documentation; and
- modify the implementation following user feedback.

Therefore, AI does not merely accelerate one phase.

It potentially interacts with **multiple phases simultaneously**.

A methodology designed specifically for this environment should consequently be organized around a

principle that can operate across those activities.

AIDF uses **intent** as that organizing principle.

Proposed AIDF Governance Layer

Because AI-generated software can introduce errors and security risks, AIDF includes a governance layer across the complete lifecycle.

The governance layer contains five controls:

Traceability

Every significant AI-generated change should be traceable to:

- a requirement;
- a user intent;
- a defect;
- a test;
- or an approved change request.

Validation

AI-generated artifacts should not automatically be accepted.

Security

Generated code and dependencies should undergo security analysis.

Human accountability

A responsible human developer or team must remain accountable for production decisions.

Feedback monitoring

Production behavior should influence future development decisions.

Proposed Measurement Model

The framework should not be considered empirically validated until it is tested against established development approaches.

A future experimental study can compare AIDF with Agile or Agile-DevOps development teams using controlled software projects.

Potential dependent variables include:

Development efficiency

- development time;
- task completion time;

- deployment frequency;
- change lead time.

Software quality

- defect density;
- escaped defects;
- test coverage;
- code quality;
- maintainability.

Adaptability

- time required to accommodate changing requirements;
- number of successful requirement changes;
- rework effort.

Security

- vulnerabilities detected before deployment;
- vulnerabilities discovered after deployment;
- security review effort.

Human factors

- developer workload;
- developer satisfaction;
- perceived usefulness of AI;
- trust in AI-generated output.

Product outcomes

- user satisfaction;
- system reliability;
- performance;
- business objective achievement.

The framework should therefore be evaluated empirically rather than accepted solely on conceptual arguments.

Research Propositions

The proposed framework generates several testable propositions.

P1: AIDF can reduce the time required to implement changing requirements compared with a predominantly phase-oriented development process.

P2: AI-assisted development within AIDF can reduce manual implementation effort while increasing the importance of verification activities.

P3: Continuous intent-to-feedback loops can reduce the amount of rework caused by late discovery of requirement changes.

P4: AIDF's effectiveness is positively associated with automated testing, version control, quality engineering and organizational AI governance.

P5: Excessive reliance on AI-generated artifacts without human validation may reduce software quality and therefore weaken AIDF outcomes.

P6: The effectiveness of AIDF depends not only on AI capability but also on the maturity of the surrounding software engineering system.

DISCUSSION

The proposed framework reflects a broader transition occurring in software engineering.

The traditional software engineer primarily transformed detailed specifications into executable software through manual development activities.

The AI-assisted engineer increasingly works with systems capable of translating higher-level intentions into candidate software artifacts.

This does not eliminate software engineering expertise.

Instead, it changes where that expertise is applied.

Architecture, requirements reasoning, security, verification, system thinking and responsible decision-making become increasingly important.

The proposed framework therefore views AI as a development participant rather than simply a development tool.

This distinction is important.

Adding an AI coding assistant to an existing methodology does not automatically create a new software development methodology. A genuine methodological change requires changes to the way development work is organized, controlled and evaluated.

AIDF proposes such a change by making evolving intent the central object of the development process.

Practical Example

Consider a university developing an AI-supported student management application.

The initial intent is:

"Provide students with a mobile system for accessing attendance, academic schedules and course information."

Under a traditional approach, the team may develop a detailed requirements document, architecture, implementation and testing plan.

Under AIDF, the initial intent is analyzed using AI-assisted tools.

The system proposes:

- application modules;
- database structures;
- APIs;
- authentication mechanisms;
- interface designs;
- security requirements; and
- test scenarios.

Developers review and approve the proposed architecture.

The AI then assists with implementation and testing.

After deployment, students report that attendance information is difficult to understand.

This feedback modifies the original intent.

The team does not need to restart the entire lifecycle.

Instead, the new intent enters the adaptive loop:

Feedback → Intent refinement → AI analysis → Implementation → Validation → Release.

This illustrates the central characteristic of AIDF: the software evolves continuously in response to evidence.

LIMITATIONS

This paper presents a conceptual framework and does not claim empirical validation.

Several limitations therefore remain.

First, the effectiveness of AIDF has not yet been demonstrated through controlled industrial experiments.

Second, AI capabilities vary considerably between tools and models.

Third, organizations differ in engineering maturity, infrastructure and governance.

Fourth, AI-generated software may introduce security, correctness and maintainability risks.

Fifth, the proposed framework may overlap with emerging AI-native software development approaches.

Consequently, future research must establish precisely where AIDF differs from existing approaches through systematic comparison and empirical evaluation.

Future Research

Future research should develop an operational version of AIDF and evaluate it through empirical software engineering studies.

A possible research programme can contain three phases.

Phase 1: Framework validation

Experts in software engineering, DevOps, AI and project management evaluate the framework using a structured Delphi study.

Phase 2: Controlled experiment

Student or professional development teams complete equivalent software projects using:

- Agile;
- Agile + AI assistance; and
- AIDF.

Performance can then be compared using predefined metrics.

Phase 3: Industrial case study

Organizations using AI-assisted development can implement AIDF for selected projects and measure outcomes over several development cycles.

This research would transform AIDF from a conceptual proposal into an empirically evaluated methodology.

CONCLUSION

Software development methodologies have continuously evolved in response to changes in technology and development practices. Waterfall introduced structured sequential development, Spiral emphasized risk-driven iteration, Agile promoted adaptability and collaboration, and DevOps connected development with continuous operational delivery.

Generative artificial intelligence introduces another major change.

AI can participate in requirements interpretation, programming, testing, documentation, debugging and software analysis. Consequently, development activities that were previously separated into relatively distinct phases can increasingly occur within a continuous human-AI interaction.

This paper proposed the Adaptive Intent-Driven Framework (AIDF) as a conceptual methodology for this environment.

AIDF does not seek to eliminate existing software engineering methodologies. Instead, it proposes a different organizing principle: **evolving intent**.

The proposed lifecycle can be summarized as:

Intent → Intelligence → Adaptive Planning → Construction → Validation → Release → Feedback → Evolving Intent

The framework places AI at the center of development assistance while retaining human responsibility for architecture, validation, security, ethics and final decisions.

The central proposition of this paper is therefore not that traditional software development has become irrelevant. Rather, it is that the emergence of AI provides an opportunity to reconsider the fundamental unit around which software development is organized.

Future empirical research is required to determine whether intent-driven, continuously adaptive development can provide measurable improvements in development efficiency, adaptability, software quality and user outcomes.

AIDF should therefore be regarded as a **researchable methodological proposal**, providing a foundation for future experimental and industrial validation.

REFERENCES

1. Altman, I., & Taylor, D. A. (1973). Social penetration: The development of interpersonal relationships. Holt, Rinehart and Winston.
2. Beck, K., et al. (2001). Manifesto for Agile Software Development.
3. Boehm, B. W. (1988). A spiral model of software development and enhancement. Computer, 21(5), 61–72.
4. DORA. (2025). State of AI-assisted Software Development. Google Cloud DORA Research.
5. DORA. (2026). Impact of Generative AI in Software Development. Google Cloud DORA Research.
6. DORA. (2025). DORA AI Capabilities Model. Google Cloud.
7. Royce, W. W. (1970). Managing the development of large software systems. Proceedings of IEEE WESCON, 1–9.
8. Treude, C., & Storey, M.-A. (2025). Generative AI and empirical software engineering: A paradigm shift.
9. SP, R. (2025). AI-Driven Development Lifecycle: Reimagining Software Engineering. AWS Builder Center.