AI in software development lifecycle

Engineering best practices for delivering production software in financial services

Artificial intelligence is already changing how software is delivered in financial services. Faster code is only one part of the story. As AI accelerates implementation, the real challenge is shifting to the rest of the software delivery lifecycle: defining the right intent, validating the output, understanding architectural decisions and ensuring that software is safe to release into production.

In traditional delivery models, progress was often measured through tickets closed, lines committed, features shipped and the quality of code review. Today, in a human-led, AI-assisted delivery model, tools can generate implementation options, tests, documentation and working code faster than teams can absorb through conventional review processes. This has moved the bottleneck to where the constraint is increasingly determined by how clearly requirements are defined, how well architectural decisions are understood and how safely software is promoted into production.

However, AI also increases the cost of weak engineering discipline. If requirements are vague, reviews are rushed or production controls are bypassed, teams can scale mistakes just as quickly as they scale delivery. In practice, that can mean flawed tests that validate broken behavior, changes that appear correct but violate business rules or production incidents caused by missing guardrails and incomplete context.

That shift matters in financial services where delivery teams work inside environments shaped by regulation, information security, data privacy, auditability, traceability and operational resilience. Faster output only creates value if it remains explainable, testable, secure, governable and supportable in production.

This paper focuses on the status quo. We describe how AI is affecting the software development lifecycle (SDLC) today, how the work of engineering teams is changing, and what we are learning from delivering production code with clients in regulated environments.

SDLC in today's financial services

Financial services firms increasingly operate as technology-led businesses. Banking, payments, wealth and asset management and operational platforms are now core channels through which customers, advisors, operations teams and regulators experience the institution. The ability to respond to business changes is therefore directly linked to the ability to implement technology change safely.

This delivery environment is optimized for controlled speed: the ability to move quickly while preserving security, traceability, auditability and production resilience. Besides feature requests, engineers are expected to also understand the workflow it supports, the data it touches, the controls it must satisfy, the customer outcome it enables and the operational consequences if it fails.

The SDLC has already evolved through agile and DevOps practices. Iteration cycles have shortened, business and technology teams have moved closer together, and automated pipelines have reduced manual handoffs. AI enters this environment as another acceleration layer. It can compress discovery, implementation, testing and documentation work, but it must operate within the same governance expectations that already define financial services delivery.

The practical question is therefore not whether AI can help teams move faster. The question is how engineering teams adapt today's ways of working, so that increased output still becomes production-quality software.

Key point: In financial services, AI must operate within a delivery model built for controlled speed, strong governance and production resilience.

Where AI is already affecting SDLC

AI is often introduced as a coding accelerator since coding tools can generate boilerplate code, explain unfamiliar code, propose refactors, create implementation variants and help engineers move from ideation to working output more quickly. However, the real impact is visible across the full lifecycle.

In requirements and analysis, AI can help draft user stories, acceptance criteria, workflow descriptions and backlog items. This is useful, but it also exposes a weakness. Human teams often compensate for incomplete requirements through shared context and informal knowledge. AI cannot be relied upon to fill those gaps consistently. Vague requirements can produce code that works in a narrow demo but fails under real business, regulatory, data or operational conditions. As a result, teams are rediscovering the importance of structured requirements, decision tables and precise acceptance criteria.

In engineering execution, AI helps teams move faster. Developers can now produce code, tests, documentation and alternative solution options in significantly less time than before. This shortens feedback loops and can help less experienced engineers contribute earlier. However, more output

also creates a new challenge: teams must be able to absorb and validate it. Code does not ship itself. It still needs to fit the architecture, meet security standards, be backed by test evidence, use data correctly and be ready for production.

In code review, AI can act as an additional quality layer in the CI/CD pipeline. It should not replace human review or become an approval gate, but it can provide a useful first pass before another engineer reviews the change. AI-assisted review can flag potential bugs, highlight unclear logic, suggest improvements, identify missing tests and catch issues that might otherwise slow down human reviewers. Used this way, AI helps improve the quality of a pull request earlier in the process while preserving human accountability for final review and approval.

In testing and validation, AI is both powerful and risky. It can generate unit tests, test data, test scaffolding and regression scenarios quickly. However, it can also create false confidence when the tests simply confirm the same flawed assumption that produced the implementation. In an AI-assisted SDLC, the test suite becomes a critical expression of intended behavior. Good tests must challenge the solution, not merely validate that generated code runs.

This is where poor-AI adoption becomes risky. AI can generate broken tests that appear to validate broken code, creating false confidence instead of real assurance. A loosely specified feature can also work in a demo but fail in production because rate limits, control logic, edge cases or operational constraints were never made explicit.

In production support and operations, AI can help summarize logs, incidents, tickets and system behavior. It can reduce the time required to gather context during triage. Yet production incidents are often solved through institutional knowledge: historical trade-offs, undocumented edge cases, batch timing, data quirks, downstream dependencies or patterns that senior engineers recognize from experience. AI can assist with this work only when that context is captured and made available. For example, an AI assistant may recommend restarting a failing service because that is the obvious textbook response, while missing the real root cause in an undocumented dependency, batch process or environment-specific failure pattern.

Key point: The biggest present-day impact of AI is not that it writes code faster. It exposes requirements, review, testing, governance and production support as the new constraints.

Bottleneck has moved - from coding to supervision

The central engineering shift is that work is moving from execution to supervision. The old bottleneck was often implementation capacity: how quickly developers could translate requirements into working code. AI reduces that bottleneck. The new bottleneck is the team's ability to define intent, decompose problems, validate output and maintain accountability for production quality.

This creates a supervisory layer of engineering work. It includes breaking problems into tasks that an AI tool can handle safely, writing prompts and specifications that cannot be easily misinterpreted, deciding where the tool can run independently and where human intervention is required, and reviewing generated output for architectural consistency rather than only syntactic correctness.

Traditional code review remains important, but it cannot carry the full burden alone when code volume increases dramatically. Before large changes are generated, engineers should review the specification, intended system behavior, functional and non-functional requirements, constraints, edge cases, test strategy, data assumptions, architectural decisions and applicable coding standards. After code generation, the review should focus on correctness, maintainability, security, traceability and fit with the broader system.

This is why specifications are becoming more valuable. A strong specification does not merely describe what to build. It must describe the expected functional behavior, the relevant non-functional requirements, the architectural constraints, applicable coding standards, data assumptions, failure modes, acceptance criteria and expected evidence. These specifications need to be precise enough for AI tools to act on, but still clear enough for humans to review, challenge and maintain. In many AI-assisted workflows, the specification and the test suite become robust product artifacts. The generated code is important, but it is no longer the only or even primary expression of engineering intent.

The project context also needs to evolve continuously. Learnings from pull request reviews, production fixes, recurring defects, architectural decisions and delivery retrospectives should feed back into the team's standards and project memory. This creates a stronger foundation for future AI-assisted work. The more accurate and current the project context is, the better the quality of the agent's output. Without that feedback loop, teams risk seeing the same issues repeated because the AI tool is working from incomplete or outdated context.

The practical implication for teams is clear: do not simply ask whether engineers can write clean code. Ask whether they can write clear specifications, design tests that catch hallucinations, debug systems they did not author line by line and explain why a generated change is safe to ship.

Key point: AI changes the definition of engineering leverage. The highest-value work is increasingly the ability to specify, supervise, validate and operate software safely.

Impact on engineering teams

AI-assisted delivery changes what strong engineering performance looks like. Teams still need implementation skills, but they also increasingly need judgment, context management, validation discipline and architectural reasoning.

Senior engineers are being pulled toward system supervision. They understand the architecture, the production history, the non-obvious dependencies and the trade-offs behind previous decisions. As AI-generated output increases, they can quickly become overloaded by review demands. If the process is not redesigned, the most experienced engineers risk becoming permanent traffic controllers rather than strategic builders. The better pattern is to involve them earlier in the lifecycle, where they can shape specifications, review architectural decisions, define guardrails and establish validation standards before large volumes of code are produced.

Junior engineers can often adapt quickly to AI-supported workflows. With the right guardrails, they can generate useful code, explore unfamiliar areas and contribute earlier than in traditional models. The risk is insufficient depth of comprehension. If junior engineers modify systems without enough experience, context and mentoring, they may become productive in the short term while weakening long-term supportability. Mentoring, code walkthroughs, test discipline and production exposure remain essential.

Mid-level engineers may face the hardest transition. They often have established habits from pre-AI delivery but may not yet have the broad architectural context that senior engineers use to supervise generated output. Their development path now needs to emphasize specification quality, architectural trade-off reasoning, test design, secure delivery practices and responsible use of AI tools.

At the team level, role boundaries are becoming more fluid. Strong delivery increasingly depends on collaboration across engineering, architecture, product, business analysis, security and domain experts. This also changes hiring and skill expectations. Financial services firms will need engineers who combine implementation ability with specification writing, validation discipline, domain understanding, controls awareness and responsible use of AI tools. The emerging pattern is engineers spending less time on repetitive implementation and more time on problem framing, validation and business value.

Key point: AI is changing what strong engineering looks like: less repetitive implementation, more judgment, validation, context and architectural reasoning.

How we deliver production code with clients

In client environments, especially in financial services, delivery must start with the client's approved tooling, information security policies, data constraints, SDLC controls and release governance.

The first delivery principle is to take an end-to-end view. Optimizing only the coding phase can create downstream congestion in review, testing, compliance and production readiness. Effective teams look across requirements, implementation, validation, documentation, release controls and support. The goal is to improve controlled throughput across the full SDLC.

The second principle is to preserve stability in production. AI-generated artifacts should still pass through secure-by-design practices, peer code review, automated quality gates, architectural decision review, traceability checks and deployment controls. AI tools can help identify issues, suggest improvements and accelerate analysis, but their output should never be trusted blindly or treated as approval-grade. Human peer review remains essential because another engineer brings fresh perspective, system context and professional judgment that the original developer or AI tool may miss. In financial services, where defects can have regulatory, security, customer, and operational consequences, this additional layer of review is not optional. AI tools should support engineers by accelerating context gathering, analysis and artifact generation while leaving accountability with the delivery team. They should not receive uncontrolled access to production systems.

The third principle is to select tooling for the delivery context. Some clients may standardize on one platform. Others may need a small set of tools across requirements, coding, testing, documentation and support. The important point is coherence. Too many disconnected tools can fragment workflows, complicate governance and create more operational burden than value. Tooling should simplify delivery, make controls easier to evidence and fit the way teams already ship software.

The fourth principle is to use accelerators where they improve the lifecycle, not only where they make developers code faster. AI-assisted requirements generation, test acceleration, backlog refinement, documentation support and production triage can all reduce friction. Capabilities such as requirements-focused and testing-focused accelerators can help teams create better upstream artifacts and stronger validation evidence, which is often more valuable than simply generating more code.

The fifth principle is to capture institutional knowledge. Every production incident, architectural trade-off, recurring defect or non-obvious operational dependency should improve the shared knowledge base. Teams should document not only what happened, but why it happened, how it was diagnosed, what assumptions mattered and what a senior engineer knew that was not obvious from the formal documentation. This improves onboarding, production resilience and the usefulness of AI-assisted support.

A final delivery lesson is that AI must be introduced with guardrails from the start. When teams apply AI without strong requirements, peer review, project context and production controls, they may ship errors faster rather than deliver value faster. In financial services, the cost of

those mistakes can be high: incorrect customer outcomes, weak audit evidence, unstable releases or operational incidents caused by changes that looked plausible but were not fully understood.

This is why successful AI adoption is not just about tool access. It is about combining acceleration with control.

Key point: Successful AI delivery with clients depends on combining acceleration with guardrails, peer review, approved tooling and strong production controls.

Engineering best practices for AI in today's SDLC

Move rigor upstream. Invest in clear requirements, structured specifications, decision tables, state models, data rules and explicit acceptance criteria before generating large volumes of code.

Treat tests as a first-class product artifact. The test suite should define expected behavior, cover edge cases, challenge assumptions and provide evidence that generated output is safe to release.

Review intent and architecture, not only code. Require architectural reasoning, design decisions, security implications, data assumptions and operational impacts to be visible before a change is merged.

Control tool access and data exposure. Use only approved tools, approved data contexts, and approved integration patterns. Do not allow AI adoption to bypass information security or regulatory obligations.

Maintain human accountability. AI can assist with analysis, generation, summarization and validation, but engineers remain accountable for correctness, security, compliance, maintainability and supportability.

Protect production. Keep deployment controls, release approvals, monitoring, incident processes, runbooks and rollback procedures intact. AI should strengthen production discipline, not weaken it.

Capture operational context. Runbooks, incident retrospectives, architectural decisions and tribal knowledge are essential inputs for safe AI-assisted delivery and support.

Upskill teams. Engineers need practical training in specification writing, context engineering, AI-assisted testing, secure delivery, code comprehension and responsible tool usage. Upskilling should be embedded in delivery work, not treated as separate theoretical exercise.

Conclusion

AI is already changing the SDLC in financial services, but the change is often misunderstood. The objective is controlled acceleration: faster delivery with clearer specifications, stronger validation, better traceability and resilient production controls.

Engineering teams are applying more effort to requirements clarity, architectural oversight, test design, governance, production readiness and institutional knowledge. Senior engineers are becoming supervisors of system intent and delivery risk. Junior engineers can contribute earlier but still need supervision and guardrails. Mid-level engineers must expand from implementation fluency into validation and trade-off reasoning.

For organizations delivering production software with clients, the practical path is to use AI across the full lifecycle while preserving the controls that make financial services software safe to ship. The promise of AI in the SDLC is not just faster code. It is better delivery discipline applied in new places.

Looking ahead, AI-assisted software delivery is likely to become more orchestrated. Instead of using individual tools for isolated tasks, engineering teams will increasingly coordinate multiple AI capabilities across the SDLC. In financial services, the differentiator will not be how much autonomy these tools have, but how safely they are embedded into governed delivery models.

Teams that invest today in clear specifications, reusable project context, strong test evidence, human review and production controls will be better positioned to scale AI responsibly tomorrow. The direction is clear: AI tools will increasingly support and accelerate execution tasks, while engineering accountability will move further toward orchestration, judgment and controlled delivery.

How Capco can help

Capco helps financial services organizations move AI-assisted software delivery from isolated experiments to governed, production-ready ways of working. We combine deep industry expertise with over 25 years of practical experience in engineering transformation, architecture, testing, data governance, regulatory change and operational resilience.

Our teams help clients assess SDLC maturity, identify high-value AI use cases, select approved tools, define responsible usage patterns and establish the guardrails needed for safe adoption. We also coach delivery teams as they apply these practices in real projects. This gives clients a practical path to improve speed and quality while maintaining the controls, traceability and accountability required in regulated environments.

Contact us to discuss how we can help you along this journey.

Authors

Vladimir Ignjatovic
Managing Principal
vladimir.ignjatovic@capco.com

Sebastian Ehrig
Senior Consultant
sebastian.ehrig@capco.com

Janusz Twardak
Senior Consultant
janusz.twardak@capco.com

Contact

Francois Melin
Partner
francois.melin@capco.com

About Capco

Capco, a Wipro company, is a global technology and management consultancy shaping change in the financial services and energy industries. For almost 30 years, we have been trusted to help our clients adapt, transform and create long-term value across capital markets, banking, payments, insurance, wealth and asset management, and the energy and utilities sectors. Our deep industry expertise, partnership mindset and award-winning Be Yourself At Work culture are amplified by our strengths in advisory, technology, data and AI innovation. We support our clients to establish clear priorities, connect vision to value, and deliver measurable impact when the stakes are highest.

Expert-led, AI-infused, impact-focused – we do not just respond to change, we help shape it. To learn more, visit www.capco.com or follow us on LinkedIn, Instagram, Facebook, and YouTube.

Worldwide Offices

APAC
Bengaluru – Electronic City
Bengaluru – Sarjapur Road
Bangkok
Chennai
Gurugram
Hong Kong
Hyderabad
Kuala Lumpur
Mumbai
Pune
Singapore

Middle East
Dubai

Europe
Berlin
Bratislava
Brussels
Edinburgh
Frankfurt
Geneva
Glasgow
London
Milan
Munich
Paris
Vienna
Warsaw
Zurich

North America
Charlotte
Chicago
Dallas
Houston
Montreal
New York
Orlando
Toronto

South America
Rio de Janeiro
São Paulo

capco.com

CAPCO
a **wipro** company

JN_7741