altimi

E-BOOK · AI ENGINEERING 2026

AI SDLC: How to effectively embed AI into your software development lifecycle

A practical guide for engineering teams, architects, and technology leaders building software in the era of large language models and AI agents.

7
SDLC phases covered

12
Agentic coding patterns

30
days to first AI sprint

altimi.com · contact@altimi.com

Wrocław, 2026

FOREWORD

Why AI SDLC is not a trend – it is the new normal

From experiment to engineering infrastructure – what changed in 2026 and what to do before you fall behind.

For the past three years, the conversation around AI in software development focused almost entirely on autocomplete and snippet generation. In 2026, that description no longer fits reality. **AI is a full participant in the software development lifecycle** – from requirements gathering and architecture design through implementation and testing to production monitoring.

Companies that adopted AI only as a developer tool are not seeing the same results as those that rethought the entire process. The distinction is fundamental: arming a developer with Copilot is one thing; building an organisation where AI operates at every SDLC layer – with proper oversight, governance, and measurable outcomes – is another.

This e-book is built on Altimi's experience across more than 350 projects delivered for companies across Western Europe. It shows **concrete patterns**, tools, and pitfalls – not academic theory. It is addressed to CTOs, senior architects, engineering leads, and anyone with real influence over how software gets built.

> **The biggest mistake we see with clients is treating AI in SDLC like a new IDE. It is not a tool – it is a change of operating model. Companies that understand this reduce lead time by 40–60% within the first quarter of implementation.**
>
> **Miłosz Cupiał**, Head of Delivery, Altimi

Most CTOs we speak to are not asking *whether* to adopt AI in their SDLC. They are asking whether they are already too late – and whether their current codebase is ready. Both fears are legitimate. Both have answers. That is what this guide is for.

In the chapters that follow, we will walk through every SDLC phase, show where AI delivers the most value, where it requires oversight, and how to build governance that protects without stifling innovation. Let us start from the foundations.

TABLE OF CONTENTS

What you will find in this e-book

NOTE FROM THE AUTHORS

This e-book is updated quarterly to reflect evolving AI tools and patterns. The version you are reading is from **Q2 2026**. You can always download the latest version at **altimi.com**.

01

CHAPTER 01

AI SDLC the new operating model

What separates AI-Assisted from AI-Native SDLC – and why that difference determines the scale of returns.

AI SDLC – the new operating model

Two approaches that separate fast-growing companies from those standing still.

Engineering literature today distinguishes two models. **AI-Assisted SDLC** – the developer leads while AI assists at selected stages (autocomplete, code review, testing). **AI-Native SDLC** – AI participates in every phase, and the engineer's role shifts toward architecture, validation, and agent oversight.

AI-ASSISTED

Developer leads

AI as a tool embedded in the IDE. Autocomplete, code generation, tests. Architecture decisions and deployment remain in human hands. Low adoption risk, limited scale benefits.

AI-NATIVE

AI as a participant

AI agents operate across the full SDLC: planning, implementation, testing, deployment, monitoring. Engineers act as architects and orchestrators. High return – requires mature governance.

Where you stand now – maturity map

L1	**Experiment**	Individual developers use Copilot or ChatGPT ad hoc. No standards, policies, or metrics in place.
L2	**Standardisation**	AI Acceptable Use Policy implemented. Approved tooling selected. AI active in code review and testing.
L3	**Integration**	AI integrated into CI/CD, PR pipeline, and documentation. First agents running in selected teams.
L4	**AI-Native**	Agents handle full implementation cycles. Engineers validate and oversee. Lead time reduced by 40–60%.

02

CHAPTER 02

Planning and requirements with AI

How AI transforms stakeholder conversations into precise user stories, acceptance criteria, and architectural proposals – faster than ever before.

Planning and requirements with AI

The requirements phase is where AI delivers measurable results fastest – provided it is used structurally.

According to PwC's 2026 analysis, **planning and ideation is the phase most strongly transformed by GenAI** (~57% of time available for optimisation). Language models are highly effective at processing unstructured stakeholder conversations into concrete artefacts: user stories, acceptance criteria, risk maps. The key condition: the prompt must carry domain context and a defined output format.

PATTERN

Structured prompt for requirements

Persona: Senior product analyst

Context: B2B SaaS application, mid-market segment, stack: React + Node.js + PostgreSQL

Task: Based on the meeting notes, generate 5 user stories with acceptance criteria in BDD format (Given / When / Then)

Format: Markdown, each story as H3, max 3 criteria per story

WHY STRUCTURE WORKS

Persona calibrates detail level and language register of the response.

Context eliminates generic answers – the model knows which system it is writing for.

Task narrows scope to a single artefact. Smaller scope, higher quality output.

Format means the result is ready to paste into Jira or Notion without post-processing.

> **We see that clients who started using AI for requirements stopped wasting time transcribing meeting notes. Their business analysts now focus on validation and prioritisation – where human judgement is genuinely irreplaceable.**

Miłosz Cupiał, Head of Delivery, Altimi

03

CHAPTER 03

Architecture design with a language model

How AI accelerates architectural decision-making without replacing the engineer's judgement – and where the boundary lies.

CHAPTER 03

Architecture design with a language model

AI as a thinking partner in architecture – not a decision-maker, but a powerful accelerator of structured reasoning.

Language models excel at pattern recognition across large code corpora. In architecture work, they are most useful as a **sparring partner**: surfacing trade-offs, proposing alternative patterns, and identifying blind spots in a proposed design. The architectural decision itself must remain with a human – AI does not carry accountability.

WHERE AI ADDS VALUE

Architecture acceleration

Generating ADR drafts (Architecture Decision Records), comparing patterns (microservices vs modular monolith), identifying potential scalability bottlenecks in a proposed design, producing C4 diagram drafts from prose descriptions.

WHERE HUMAN STAYS IN CHARGE

Architectural judgement

Final technology selection, trade-off decisions involving business context, security architecture with compliance implications, cost modelling under real operational constraints, and all decisions that carry organisational accountability.

AGENTS.md – the foundation for consistent AI output

One of the highest-leverage practices in 2026 is creating a project-level context file – typically named **AGENTS.md** or **CLAUDE.md** – that documents conventions, architectural decisions, and design patterns used in the repository. Without it, agents generate inconsistent code and introduce technical debt with every iteration.

01	**Tech stack**	Languages, frameworks, libraries in use. Approved and forbidden dependencies. Version constraints.
02	**Patterns**	Naming conventions, module structure, error handling patterns, logging standards.
03	**Decisions log**	Key architectural decisions (ADRs) with rationale. This prevents agents from re-opening settled debates.
04	**Security rules**	What must never be sent to external models, authentication patterns required, secrets management approach.

04

CHAPTER 04

Agentic coding – implementation patterns

Which agent patterns work in production in 2026, how to deploy them, and what to avoid when scaling.

CHAPTER 04

Agentic coding – implementation patterns

In 2026, AI agents operate in production SDLC – but only in specific shapes, at specific stages, with specific integration patterns.

Deploying agents across every phase simultaneously is a recipe for chaos. Practice shows that **the best results come from teams that start with one pattern, one phase, and three metrics**. Below are four proven patterns – each deployable within 2–4 weeks.

PATTERN 01

PR Drafter

Agent analyses branch diff and generates a pull request draft – description, checklist, linked tickets. **Saving:** 20–30 min per PR, eliminates context-free "empty" PRs.

PATTERN 02

Test Scaffolder

Agent analyses new PRs, identifies missing test coverage, generates unit test scaffolds. **Requirement:** test conventions documented in AGENTS.md or CLAUDE.md.

PATTERN 03

On-call Summariser

Agent monitors logs and alerts, groups incidents, generates root cause summaries and remediation steps. **Result:** MTTR reduced by up to 35% in the first month.

PATTERN 04

Knowledge Agent

RAG-based agent answers questions about code, architecture, and design decisions. Fed from Confluence, ADRs, and repo comments. Available in IDE and Slack.

> **We started with the PR Drafter on one team. After two weeks, developers stopped turning it off – that is the best adoption signal you can get. Only then did we roll out the Test Scaffolder and begin measuring coverage. That is what healthy agent adoption looks like – step by step, not big bang.**
>
> Senior QA Engineer, Altimi

TDD and code quality as the foundation of AI-readiness

AI works best where code is healthy. Healthy code is not an accident – it is an architectural decision.

Research from CodeScene shows that **AI agents make mistakes on the same patterns as humans** – and in degraded code, they amplify those mistakes. Before introducing agentic coding, measure the Code Health of your codebase and resolve the highest-risk hotspots. The recommended threshold is 9.5–10.0/10.

TDD First – the golden rule of AI coding

01	**Write the test**	Before sending a prompt to an agent, write the test (unit or integration). The test defines the contract – it is the specification, not the prompt.
02	**Give the agent the test**	Pass the test file to the agent and ask for an implementation that satisfies it. AI has a concrete goal – it does not interpret what "should work."
03	**Review the code**	Do not accept code just because tests pass. Check readability, alignment with project conventions, and the absence of unintended side effects.
04	**Merge and measure**	Merge with automated coverage measurement. A coverage regression signals that AI is generating code without tests – fix it with a prompt correction.

05

CHAPTER 05

Testing and quality assurance with AI

AI in QA does not replace testers – it shifts their attention to what genuinely requires human judgement: edge cases, UX, and security.

Testing and quality assurance with AI

Where AI in QA delivers real efficiency gains – and where human oversight is non-negotiable.

QA area	AI delivers value	Requires human oversight
Unit tests	Scaffolding, happy-path case generation	Edge cases, domain-specific logic
Integration tests	Mock generation, data schema creation	Service dependencies, call sequencing
E2E / UI	Flow suggestions, Cucumber step generation	UX validation, accessibility, responsiveness
Security	SAST, vulnerability scanning, fix suggestions	Threat modelling, penetration testing
Regression	Automated execution, diff reporting	Criticality assessment, merge-block decisions

> **As a tester, AI has taken the burden of writing dozens of baseline scenarios off my plate. Now I can focus on what is actually interesting: what will NOT work in edge cases, how a real user behaves with the system. The quality of our regression suite went up – because we have more time to think, not to type.**
>
> Senior QA Engineer, Altimi

The critical practice is maintaining **code coverage as a behavioural guardrail**. When AI generates code at high volume, the absence of an automated coverage gate in the CI/CD pipeline means regressions reach production faster than ever before. Minimum branch coverage enforced as a merge condition is low risk, high value.

06

CHAPTER 06

Security and governance of AI-generated code

Without governance, AI does not create value – it creates risk. How to build a framework that protects without stifling innovation.

Security and governance of AI-generated code

Three governance pillars every organisation must implement before scaling AI in SDLC.

PILLAR 01

Acceptable Use Policy

A clear policy: which data may reach public models, which tools are approved, who is accountable for AI code review. No policy = shadow AI.

PILLAR 02

Automated scanning

Automated scanning of every PR for vulnerabilities (AI SAST). Does not replace human review – operates as the first security layer.

PILLAR 03

AGENTS.md

A file in the repository documenting conventions, architectural patterns, and project rules. Without it, agents act inconsistently and generate technical debt.

What we never send to a model

Absolute prohibitions, regardless of tool or vendor data policy:

✗	**API keys and passwords**	No key, token, secret, or credentials – even in a code comment. Use environment variables and a secrets vault.
✗	**Customer PII**	Personal data, email addresses, card numbers, national IDs. Always work with anonymised test data.
✗	**Intellectual property**	Unpublished proprietary client code does not go to public models. Use local models or privacy-guaranteed API endpoints.

07

CHAPTER 07

Legacy modernization with AI

How to prepare a codebase for agents – and what it costs to skip that preparation, in numbers from real Altimi projects.

Legacy modernization with AI

Before deploying agents to a legacy system – assess AI-readiness first. Not every codebase is ready.

Legacy codebases and AI can work beautifully together – or catastrophically. **The sequence is critical:** refactor to AI-readiness first, then automate with agents. Reversing that order amplifies technical debt at a speed no human team can match.

RESULTS FROM ALTIMI PROJECTS

40%	3×	8 wks
Average lead time reduction after AI SDLC	Test coverage growth after 90 days	Time from audit to AI-readiness

Legacy modernization workflow with AI

01	**AI-readiness audit**	Code Health analysis, test coverage, dependency mapping, documentation assessment. Identify hotspots blocking agents.
02	**Priority refactoring**	Remove hotspots using the matrix: Code Health × change frequency. Target: Code Health ≥ 9.5 in the pilot module.
03	**AGENTS.md + docs**	Build agent context: conventions, patterns, decision history. Without this, agents generate inconsistent code.
04	**Agentic coding pilot**	One module, one pattern (e.g. Test Scaffolder). Measure results before scaling to additional modules.

Is your codebase ready for agents? Altimi's AI-readiness audit gives you a prioritised roadmap in under 2 weeks. Book a free scoping call →

08

CHAPTER 08

Metrics and measuring AI effectiveness in SDLC

You cannot manage what you do not measure. The three numbers every engineering leadership dashboard should track.

Metrics and measuring AI effectiveness in SDLC

You cannot manage what you do not measure. Three metric levels that belong on every CTO's dashboard.

LEVEL 1 · SPEED

Lead time

Time from backlog task to production deployment. **Target:** ≥30% reduction after 90 days of AI SDLC.

LEVEL 2 · QUALITY

Test coverage

Branch coverage measured on every PR. **Target:** ≥ 20 pp. growth after Test Scaffolder deployment.

LEVEL 3 · RELIABILITY

MTTR

Mean Time To Recovery after incidents. **Target:** ≥ 35% reduction after On-call Summariser deployment.

Metric	Before AI SDLC (baseline)	After 90 days	How to measure
Lead time	12–18 days	7–10 days	Jira / Linear deployment tracking
PR review time	4–6 hours	1.5–2.5 hours	GitHub / GitLab analytics
Test coverage	45–55%	70–80%	SonarQube / Codecov
Defect escape rate	8–12 per sprint	3–5 per sprint	Bug tracker (Jira / Linear)
MTTR	3–4 hours	1.5–2 hours	PagerDuty / OpsGenie

Want to know your baseline? We help you set up these dashboards and establish pre-AI baselines before your first sprint. See how Altimi does it →

09

CHAPTER 09

Implementation checklist – 30 days to AI SDLC

A week-by-week action plan to move from your first pilot to a measurable AI-assisted sprint.

CHAPTER 09

Implementation checklist – 30 days to AI SDLC

A minimal action plan for organisations starting their AI software development journey.

WEEK 1–2 · FOUNDATION

- [x] Run an AI-readiness audit (Code Health, test coverage, documentation)
- [x] Draft an Acceptable Use Policy for AI tools (which data, which tools)
- [x] Select one module and one pilot team
- [x] Install and configure an approved AI tool (Cursor, Copilot, Windsurf)
- [x] Create AGENTS.md or CLAUDE.md in the pilot repository

WEEK 3–4 · PILOT

- [x] Deploy PR Drafter on the pilot repository
- [x] Add automated security scanning to the CI/CD pipeline
- [x] Start measuring three baseline metrics (lead time, coverage, MTTR)
- [x] Run a retrospective after the first AI-assisted sprint
- [x] Plan the scale-out to additional modules (or adjust the pattern)

THE GOLDEN RULE

One pattern. One team. Three metrics.

Do not deploy everything at once. One successful, measured implementation delivers more than five unfinished experiments. Before you scale, validate the hypothesis with data – not with a feeling.

> **I never arrive at a client with a ready-made solution. I arrive with questions: where it hurts, what is blocking progress, what makes sense for their scale and engineering culture. Only then do we build the roadmap together. That is the only path that delivers lasting results.**

Miłosz Cupiał, Head of Delivery, Altimi

Ready to run your first AI sprint in 30 days?
Start with a free 60-minute scoping session with a senior Altimi architect.

Book free session →

10

CHAPTER 10

How Altimi supports AI transformation

Meet the team and the services behind 350+ AI SDLC projects delivered across Europe.

How Altimi supports AI transformation

From AI-readiness audit to full agentic SDLC – a structured path with measurable outcomes, backed by 20 years of engineering experience.

SERVICE

AI-readiness audit

Comprehensive codebase analysis: Code Health, test coverage, documentation, governance gaps. Deliverable: a report with an AI SDLC implementation roadmap and prioritised action list.

SERVICE

Legacy modernization

Refactoring and preparing a codebase for agent collaboration. From audit to AI-ready state in 6–10 weeks. Proven in projects for companies across DACH and Benelux.

SERVICE

Agentic SDLC setup

Deployment of agentic coding patterns, AGENTS.md configuration, CI/CD and security tooling integration. First measurable results within 30 days.

SERVICE

Technology Due Diligence

AI-readiness assessment of portfolio companies for PE/VC funds and acquirers. Identifies technological risks and hidden value before a transaction.

About the author: Miłosz Cupiał, Head of Delivery at Altimi, is an experienced delivery leader with 15+ years of experience in complex international projects across digital transformation, finance, healthcare, and e-commerce.

altimi

LET'S TALK

Your competitors are already running AI agents in production.

Book a free 60-minute session with a senior Altimi architect. We will assess your current SDLC maturity, identify the highest-impact starting points, and send you a prioritised transformation roadmap – within 48 hours.

Book free audit session →

contact@altimi.com

20
Years of engineering experience

250
Engineering experts

30
days to first AI sprint