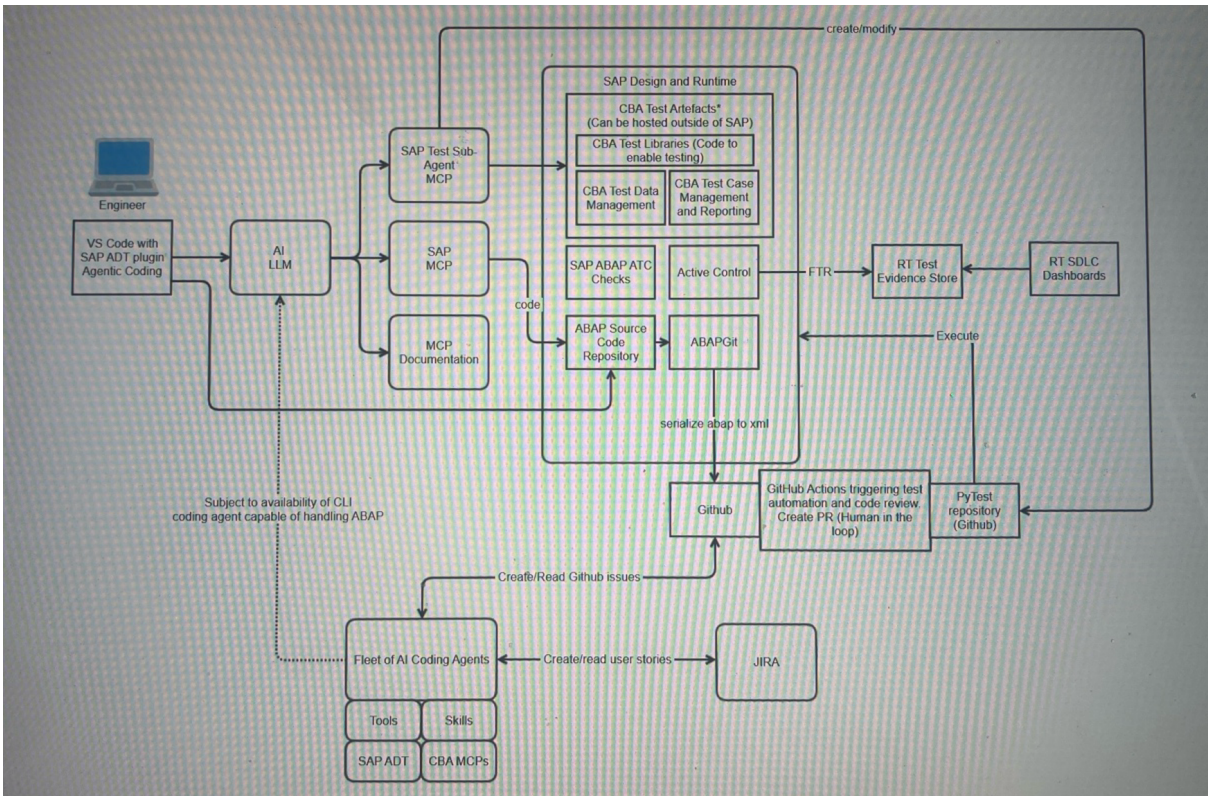

While the AI coding story continue to evolve at a ridiculous rapid pace, it is impossible to predict what the future of software engineering looks like.
This is the best "guesstimate" I can personally come up with, well, a simplied version of the dream in my head.
This is by no means gospel, and will prob be rendered irrelevant in 6 months, but let's put it down on paper for scrutiny.
Current state
Today, SAP development remains largely manual due to tooling constraints. Throughout 2025, significant effort was undertaken by Dave Beardall and Harish Ramanarayanan to evaluate a range of development tools available in the market. Currently, engineers primarily rely on SAPGUIn ageing proprietary IDE that, while powerful, is outdated-or Eclipse with the SAP ADT plugin. Eclipse can integrate with GitHub Copilot; however, Copilot frequently hallucinates in the SAP/ABAP context. Furthermore, for security reasons, CA has disabled MCP support for GitHub Copilot on Mac SOE, preventing the injection of additional context. As a result, Copilot operates with little awareness of the existing SAP codebase, often generating code that references non-existent function modules or class methods, or invents incorrect input/output parameters. This significantly limits its usefulness and renders it a sub-par assistant.
While SAP does not yet fully support VS Code for ABAP development, its AI coding ecosystem is far more mature. Extensions such as RooCode and Cline are feature-rich and can integrate with SAP MCP servers for documentation lookup, basic SAP interactions, and code reviews. However, these tools cannot directly create or deploy code into SAP systems. Developers must manually copy and paste generated code, execute tests, provide feedback, and iterate. This workflow is inefficient and far from ideal.

At best, our current state can be described as having limited "chat-coding" coding assistance, with engineers still performing the majority of the work. To be clear, AI tools can be effective in greenfield scenarios where minimal existing context is required. With a well-crafted prompt, they can generate up to 50% of usable code, assist with basic refactoring, and produce simple unit test scaffolding. For brownfield development, however, their value is minimal and often not worth the effort.
SAP has also introduced an AI coding assistant, SAP Joule, specifically trained for ABAP development. Unfortunately, it is only available on SAP BP, supports a limited set of use cases, and is not well suited to self hosted SAP environments such as ours.
At SAP TechEd 2025, SAP announced upcoming support for VS Code as an official IDE for ABAP development. This is a significant development for us, as it would allow us to finally move away from Eclipse and adopt VS Code as the primary development environment.

(Not too distant) Future State

Now, with my crystal ball, I'm imagining this is what development lifecycle will look like in the not too distant future. I classify them as two phases, but of course there are many more possibilities.

Phase 1: AI-Augmented Engineering

In this phase, humans remain the primary source of intent and decision-making. AI is used to optimise execution, compliance, and delivery throughput, with governance remaining human-led and AI-assisted.
Engineers work in VS Code with the SAP ADT extension, enabling native CRUD operations on SAP development objects. They leverage AI coding assistants (e.g. Roo, Cline, GitHub Copilot, Claude) as peer programmers to improve productivity, code quality, and development efficiency.
To ensure consistency and scalability, AI skills, tooling, and reusable assets are centrally supported-either by a dedicated enablement team or through open inner-source contribution. This approach promotes standardisation while allowing teams to innovate.
Engineers also use AI to create and modify automated test cases relevant to their changes. Deep expertise in testing frameworks is no longer a prerequisite; instead, engineers communicate test intent using natural language while ensuring changes are safe, secure, and suitable for future evolution.
To improve accuracy and contextual awareness, SAP MCP servers are made available to provide AI with rich, enterprise-specific context, including:

- SAP Banking documentation
- CBA functional and technical documentation, business requirements, and specifications
- CBA SAP coding standards and architectural guardrails
- CBA testing assistants (enabling interaction with tools such as Tosca or equivalent internal frameworks)

- Historical production incidents and root causes
- Known SAP performance anti-patterns
- Regulatory obligations mapped to affected code areas
- Environment-specific constraints (e.g. production-only restrictions)
- Legacy system quirks and "tribal knowledge" repositories
- Additional enterprise-specific knowledge sources as required

SAP ATC code-smell checks are executed against all changed code to ensure compliance with coding standards, security requirements, and performance expectations. This functions as an automated quality and risk gate.

Source code is synchronised to GitHub via ABAPGit, enabling pull requests and GitHub Actions. This supports a robust "human-in-the-loop" model, ensuring all guardrails are met and changes are reviewed for compliance, risk, and policy adherence.

Active Control is leveraged as part of the proprietary build and deployment pipeline to promote code across environments and ultimately into production. Custom enhancements enable automated test evidence to be published to the evidence store, supporting Fast Track Release capabilities. This ensures full SDLC compliance, automatically generates change records, and significantly reduces lead time to change (LTTC), allowing faster delivery of value to customers.

Documentation is automatically generated or updated as part of the workflow.

While this description is intentionally simplified, the core outcome is clear: engineers are equipped with a significantly stronger toolset that materially improves development speed and code quality. Routine and low-value activities-such as manual documentation and repetitive testing-are largely automated, allowing engineers to focus on higher-value work.

A subtle but important shift also occurs:

- Engineers increasingly review intent and outcomes rather than line-by-line mechanics
- Pull request reviews become focused on policy compliance, risk, and business impact rather than syntax and implementation details

Phase 2: AI-Augmented Delivery Systems

In this phase, human involvement shifts decisively toward intent definition, prioritisation, and risk ownership. AI agents-take on bounded delivery tasks with a high degree of autonomy, while governance evolves int a policy-driven and evidence-based model.

Extending beyond Al-assisted coding, we consider a delivery model in which autonomous Al agents actively augment engineering squads. While this may have seemed aspirational only a few years ago, it is now increasingly plausible. In 2025, CBA internally launched Project Coral, which integrates with GitHub to automatically detect and remediate code quality issues-demonstrating early, practical steps toward this future

Under this model, humans focus primarily on defining requirements, specifying outcomes, and crafting high-quality user stories in tools such as Jira. Execution then becomes increasingly agent-driven.

A Plausible Agent Lifecycle

An autonomous delivery agent could operate as follows:

- Ingest a structured work item (Jira or GitHub)
- Perform impact analysis across code, tests, data, and integrations
- Propose a solution approach with an associated risk profile
- Await human approval of the proposed approach
- Implement the change with full SDLC artefacts
- Raise a pull request with an attached evidence bundle
- Iterate based on human or automated feedback

Once approved, the change progresses automatically to the next test environment for contract and system testing. Depending on complexity, end-to-end and performance testing may be triggered. Leveraging Fast Track Release capabilities, a change record is created automatically pending approval, and deployment can occur in the next suitable release window.

Prerequisites for Autonomous Delivery

To enable this operating model, several conditions must be met:

- SAP provides ADT support for CLI-based agents, or an equivalent mechanism to update SAP code without reliance on a UI-driven IDE such as VS Code
- High-quality, current documentation of the existing system landscape
- Highly detailed user stories and issue registers that clearly define intent and acceptance criteria
- Strong automation to validate that changes do not introduce regressions or systemic risk
- A cultural shift that recognises AI agents as first-class contributors within delivery teams
- Clear decision frameworks to determine which user stories are suitable for AI execution versus human ownership (e.g. based on complexity, risk, or innovation requirements)

AI-Generated Delivery Artefacts

Autonomous agents will naturally generate structured artefacts, including:

- Change intent summaries
- Assumption logs

AI-Generated Delivery Artefacts

Autonomous agents will naturally generate structured artefacts, including:

- Change intent summaries
- Assumption logs

- Risk classifications
- Confidence scores
- Automated rollback and remediation plans

These artefacts become foundational inputs for:

- Audit and compliance
- Incident investigation and response
- Regulatory assurance and confidence

Beyond Single Agents: Agent-to-Agent Collaboration

An often-overlooked evolution is agent-to-agent collaboration. Future delivery squads may include specialised agents, such as:

- Coding agents
- Testing agents
- Performance agents
- Security agents

These agents challenge each other's outputs, generate adversarial test scenarios, and escalate uncertainty or ambiguity to human engineers when confidence thresholds are not met.

The Evolving Role of the Engineer

As this model matures, the engineer's role evolves toward system design, architectural oversight, review, and risk ownership. Engineers become stewards of intent, safety, and quality rather than executors o implementation detail.

This represents a fundamental operating-model shift—not merely the adoption of new tools, but a redefinition of how software is designed, delivered, governed, and trusted at scale.