Engineering Leaders Community

AI SDLC FOR SAP · 10 NOVEMBER 2026

Your AI SDLC proposal

Built from your answers: where your team is today, what to change first, how to install the tools and how your developers use each of the 10 AI skills.

7 /10 **AI reads and checks, a person writes**

Claude Code in the Eclipse terminal, connected to SAP's MCP server + arc1 for reading.

What AI can do at each step

Step	Status	How
Jira ticket → specification (AI sharpens it with you)	Partly	You paste the ticket or use an export.
Specification → IDE, AI knows your system	AI	AI reads live code, DDIC and where-used.
Build a new feature	Partly	AI proposes code, a person writes it into SAP.
New ABAP objects (classes, CDS, DDIC)	Partly	AI prepares the source, a person creates the object.
Transport through the landscape	By hand	A person handles transports.
Fix a bug from a dump	AI	AI reads the pasted dump and traces the cause in the live code.
Refactor with tests	Partly	AI proposes, a person runs the tests.
Move data logic into CDS views	Partly	AI writes the CDS view and the new code, a person creates them in SAP.
Reverse engineering, root cause	AI	AI follows the call chain and where-used in the system.
Tests, code review, documentation	AI	AI runs ATC and ABAP Unit and fixes the code from the results.

What to change first

1. Update ADT to 3.62 or newer. From 3.62 SAP's own MCP server reads and edits source for Claude Code, so you don't need a community server to write. To let AI write, turn on Enable Edit File tool and put one Z package, for example ZAI_*, on the Access Restrictions package list.
2. Give developers a read-only Jira API token. AI then reads tickets directly and works on the specification without copy-paste.

Install the full toolset

For one developer, matched to your answers, in this order.

1. Send this list to IT

IT installs software in your company. The steps below are their checklist, in this order.

2. Claude licence and Claude Code

An admin switches on Claude Code for all developers in the Claude admin settings (Team or Enterprise). Everyone logs in with the company account, never a private one: on private accounts Anthropic may use the data for training.

3. Eclipse 2026-09 with ADT

Download Eclipse IDE 2026-09 from eclipse.org. In Eclipse open Help → Install New Software, paste the address below and select ABAP Development Tools. Then File → New → ABAP Project and connect the DEV system from SAP Logon.

```
https://tools.hana.ondemand.com/2026-09
```

4. Claude Code (Windows)

Install Git for Windows, then run the installer in PowerShell and log in with the company account. Run Claude Code natively on Windows, not in WSL.

```
irm https://claude.ai/install.ps1 | iex
claude
```

5. Claude Code inside Eclipse

In Eclipse open Window → Show View → Terminal and run Claude Code there. Code, checks and AI sit in one window.

6. Switch on SAP's MCP server

Window → Preferences → ABAP Development → MCP Server → Enable ADT MCP Server. Keep Access Restrictions on and put only the DEV project on the project list. Leave the package list empty, so AI only reads. When you want writes, add one Z package. The server is free to use; SAP supports its MCP tools only with a Joule for Developers licence, which isn't offered for on-premise NetWeaver 7.x. Copy the generated token. To start the server with Eclipse, add this line to eclipse.ini after -vmargs:

```
-DadtMcpServerPrefEnabled=true
```

7. Connect Claude Code to the MCP server

In the terminal, with the token from the previous step. Never commit the token to a repository.

```
claude mcp add --transport http --scope user abap-adt http://localhost:2234/mcp --header
    "Authorization: Bearer <TOKEN>"
claude mcp list
```

8. The arc1 add-on for reading code

ADT 3.60 and 3.61 can’t hand source code to Claude Code; SAP added that in 3.62. Until you update, the community add-on arc1 adds read_source, where_used, package_contents, atc_check and run_unit_tests to the same server. Install it into Eclipse following github.com/arc-mcp/arc1-adt-abap-mcp-ext and block its arc1_sap_http_post tool (see Rules): it can post to any ADT endpoint, and SAP's Access Restrictions most likely don't stop it.

9. abapGit on DEV

In SE38 create the report ZABAPGIT_STANDALONE in the local package $ABAPGIT, paste in zabapgit_standalone.prog.abap from GitHub and activate it. To export a package: New Offline → choose the package → Export ZIP, then unzip it into the team repository.

10. Jira through an export

Save a Jira export (CSV) into the team repository, AI reads the files. Replace names and user IDs with initials first.

11. The 10 skills as a Claude Code plugin

The skills below are a free, open plugin (github.com/marian-kamenistak/sap-ai-skills). Each developer installs it with two commands in Claude Code. Copy its templates/CLAUDE.md into your team repository and fill in your release, Z/Y naming rules and what must never go to AI.

```
/plugin marketplace add marian-kamenistak/sap-ai-skills
/plugin install sap-abap@sap-ai-skills
```

12. Rules

Claude Code asks before every MCP tool call by default. Keep that for everything that writes. In the team repository’s .claude/settings.json, block arc1’s http_post (SAP’s Access Restrictions most likely don’t stop it), transport creation (transports stay with people), the short-dump feed (dumps can hold personal data) and the three tools that spend Joule tokens. With a Joule for Developers licence, delete those last three lines.

```
{
  "permissions": {
    "deny": [
      "mcp__abap-adt__arc1_sap_http_post",
      "mcp__abap-adt__abap_transport_create_request",
      "mcp__abap-adt__abap_feed_query",
      "mcp__abap-adt__abap_explain_object",
      "mcp__abap-adt__abap_atc_ai_*",
      "mcp__abap-adt__abap_docs_dcp"
    ]
  }
}
```

13. Check that it works

In Claude Code or Copilot ask “list my ABAP destinations” and “find class ZCL_... and explain what it does”. The answer must come from your DEV system, not from general knowledge.

How your developers use each skill

Ten skills cover the whole development lifecycle, numbered in the order a change goes through them. The steps match what AI is allowed to do in your setup.

`spec-grill` A specification that survives development

Input: a ticket from a Jira export or pasted text

1. AI asks the author one question at a time: business goal and who benefits, affected SAP objects (tables, function modules, BAdIs, interfaces), current and target behaviour, acceptance criteria as given/when/then, test data and edge cases, authorizations and personal data, dependencies (interfaces, jobs, transports) and what is out of scope.
2. It checks every named object in the system: does it exist, where is it used, what fields does it have.
3. It scores the specification 0 to 10 on the same rubric. Below 8 it goes back to the author with the list of what is missing.

A person decides: The author answers and decides when the specification is done.

Output: A Markdown specification with its score, ready to go back into Jira.

`reverse-engineer` How it works and why it fails

Input: a question like “how does X work” or “why does Y happen”

1. AI finds the entry point (transaction, report, BAdI) and follows the call chain and where-used across the whole system.
2. It explains the data flow and draws it.
3. For a bug, it lists the likely causes in order, each with evidence from the code.

A person decides: A person confirms the cause in the debugger.

Output: An explanation with a diagram, the input for a bugfix or a refactor.

`implement` A new feature from specification to transport

Input: an approved specification scored 8 or higher

1. AI reads the code straight from DEV over MCP, and finds the affected places and where-used.
2. It proposes two technical options with their impact and risks. A person picks one.
3. AI prepares the code, a person pastes it into ADT and activates it. AI then runs ATC and ABAP Unit over MCP, reads the results and proposes fixes.
4. It writes ABAP Unit tests for the acceptance criteria in the specification.
5. A person creates and releases the transport; AI writes its description.

A person decides: Choosing the option, reviewing the diff, releasing the transport.

Output: Activated code with tests and a short technical note for Jira.

`test-case` Tests

Input: a class, or a specification with acceptance criteria

1. AI writes ABAP Unit tests as given/when/then and replaces the database with a local test double behind an interface (CL_OSQL_TEST_ENVIRONMENT needs 7.51, TEST-SEAM 7.50).
2. It runs the tests over MCP and fixes what fails.
3. It also writes manual test scenarios for QAS.

A person decides: A person checks that the tests prove the specification, not just the code.

Output: A test class and test scenarios for QAS.

`code-review` Code review

Input: a diff or an exported package

1. AI runs ATC and uses its findings as the base.
2. It checks Clean ABAP, missing AUTHORITY-CHECK, SELECT inside a loop, SELECT *, FOR ALL ENTRIES with an empty table, dynamic SQL, performance on HANA and your naming rules.
3. Each finding gets a severity and a proposed fix.

A person decides: The author decides which findings to fix.

Output: Findings ordered by severity, as a comment on the Jira ticket.

`refactor` Refactoring

Input: old Z code

1. AI first writes characterization tests that pin down today's behaviour.
2. It refactors in small steps, following Clean ABAP.
3. After each step a person runs the tests.

A person decides: Reviewing the diff after each step.

Output: Cleaner code with the same behaviour and tests that prove it.

`cds-rewrite` Move data logic from ABAP into CDS views

Why do it: The database does the joins, filters and totals instead of the ABAP server, so less data travels and slow reports usually get faster. One CDS view then serves the report, a Fiori app and an API, carries its authorization rules (access control) with it, and follows the direction SAP builds S/4HANA in (RAP, ABAP Cloud, clean core).

Input: a report or class that reads data with SELECTs inside loops, FOR ALL ENTRIES or joins and totals built in ABAP

1. AI reads the program and lists every data access: each SELECT, joins done in loops, FOR ALL ENTRIES, totals and filters computed in ABAP, and where-used of the tables.
2. It first writes ABAP Unit tests that pin down today's output on test data (a local test double behind an interface), so the rewrite can prove it returns the same result.
3. It proposes the CDS design: which joins, filters, totals and calculated fields move into a DDIC-based CDS view with DEFINE VIEW (view entities need 7.55), which associations it exposes, what stays in ABAP (updates, complex procedural rules), and an access control that replaces scattered AUTHORITY-CHECKs where it fits. A person approves the design.
4. AI writes the CDS source and the new program code, a person creates and activates them in ADT, then AI runs the tests over MCP and fixes what fails.
5. It compares the runtime before and after with SAT or an ST05 SQL trace on realistic data and reports the numbers. No gain, no rewrite.
6. A person creates and releases the transport; AI writes its description.

A person decides: Approving the CDS design and checking the before and after numbers.

Output: A CDS view with access control and tests, a shorter program that reads from it, and the runtime before and after.

`documentation` Documentation

Input: an object or a package

1. AI reads the code straight from DEV over MCP.
2. It describes the purpose, inputs and outputs, tables used, who calls the object and what it calls.
3. It adds a sequence diagram in Mermaid and the known risks.

A person decides: The object owner confirms the documentation.

Output: Markdown for Confluence or the Jira ticket.

`transport-release` Transport through the landscape

Input: a finished change in DEV

1. AI checks the list of objects you give it.
2. It writes the transport description, a QAS test plan and a rollback plan.
3. Releasing the transport and importing it into QAS and PRD stays with people, following your process.

A person decides: Releasing the transport and the import to production.

Output: Transport description, QAS and PRD checklist, rollback plan.

`bugfix` Fix a bug

Input: the Jira bug and the ST22 dump text without personal data

1. From the dump, AI finds where it crashed and walks the call chain back to the entry point.
2. It describes the cause and shows the lines of code that prove it.
3. It first writes a test that reproduces the bug, then the smallest fix.
4. AI prepares the code, a person pastes it into ADT and activates it. AI then runs ATC and ABAP Unit over MCP, reads the results and proposes fixes.
5. A person creates and releases the transport; AI writes its description.

A person decides: A person confirms the cause before the fix and reviews the diff.

Output: The fix, a test that stops the bug coming back, and the cause written into Jira.

Your answers

Which AI tool will your ABAP developers use?
Claude (Team or Enterprise)

Is Claude Code switched on for your developers?
Yes

Where do your ABAP developers write code every day?
Eclipse with ADT 3.60 or 3.61

Which system do you develop on?
ECC on 7.50 or newer

Can Eclipse connect to your SAP system (are the ADT services switched on)?
Yes

How do changes reach production?
Transports through TMS (DEV → QAS → PRD)

Will you allow SAP's MCP server in Eclipse, against DEV only?
Yes, read and checks only

A central MCP server inside your network, behind the firewall and reached over VPN?
No

Would you build your own MCP server with only the tools you choose?
No

Can developers use open-source community tools (abapGit, arc1, community MCP servers)?
Yes

Can developers use abapGit on the development system (DEV) to copy code into files?
Export to files only

How will AI get to your Jira tickets?
Only through an export

Do you use ATC (ABAP Test Cockpit)?
Yes

Do you write ABAP Unit tests?
Sometimes

What may go to AI?
Source code (Z objects), DDIC structures, Dumps and logs without personal data

What do developers work on?

Windows

Who installs software?

IT

Retake the test: https://www.engineeringleaders.io/academy/ai-sdlc-for-sap