BREATHING NEW LIFE INTO RPG

AI as Your Modernization Partner

Common Europe IBM i Conference • 2026

Koen Decorte • CD-Invest nv

What You'll Leave With

- How AI reads and understands legacy RPG
- Live examples: fixed-format → RPG Free in seconds
- JORI BOM automation: a real-world MCP win
- Validation, CI/CD integration & best practices
- Stratum: making your codebase AI-readable
- A sustainable modernization strategy for your shop

"Your legacy code is not the problem.
Not reading it is."

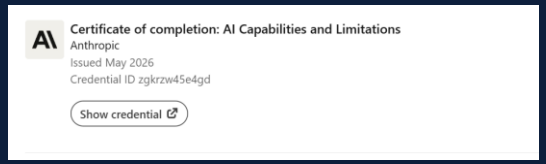
CD INVEST IT SOLUTIONS

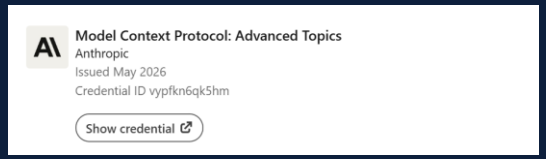
YOUR GUIDE TODAY

Koen Decorte

Senior IBM i Architect & AI Enthusiast • CD-Invest nv

- Belgian IBM i consultancy — 30+ years
- PEPPOL Access Point PBE000833
- ERP / Accounting / CRM on IBM i
- 17 IBM i case studies published
- Clients: JORI Furniture, Deknudt Frames, Winsol, …
- 9 Antrophic certifications





01 — The Legacy Burden
Why modernization can't wait

02 — AI Fundamentals
How AI reads your code

03 — The Agent Stack
Skills · MCP · CLI · Agents

04 — Live Rewrites
Fixed-format → RPG Free demos

05 — JORI Case Study
AI-powered BOM automation

06 — Validation & CI/CD
Trust, verify, ship

07 — Stratum
Your codebase, AI-readable

08 — Tool Comparison
Claude Code, IBM Bob vs the field

09 — Strategy
Building your modernization roadmap

SECTION 01

The Legacy Burden

Understanding why modernization is no longer optional

The invisible enemy in your IBM i shop

> *When the last person who understood this program retires, the business logic goes with them.*
> *We are not running code — we are running institutional memory that nobody wrote down.*

What we actually had

- Fixed-format RPG — written before SQL, APIs, or readability mattered
- Documentation nobody read or updated
- Logical files as invisible access paths — unknown field usage
- Zero documentation. Zero tests. Zero data lineage.

Why rewriting always fails

- You rewrite what you CAN see. The bugs are in what you CANNOT see.
- Edge cases accumulated over 30 years don't announce themselves
- Sentinel date 0001-01-01 means 'empty' — a rewriter would delete that logic
- The answer is not to replace the knowledge. Extract it first.

IBM i: The Invisible Powerhouse

Running quietly behind some of the world's most critical business operations

40+

years of
continuous evolution

120K+

installations
worldwide

30%

of Fortune 500
runs IBM i

99.999%

uptime target
— and it delivers

The platform is modern. Is your code?

Four Decades of RPG Evolution

The language grew — but not all code grew with it

1959	1969	1979	1994	2001	2013+
RPG I	**RPG II**	**RPG III**	**RPG IV**	**RPG Free**	**Full Free**
Report generation, punched cards	Fixed-format indicators, subroutines	Structured logic, CL integration	ILE, data structures, better types	Free-format, modern syntax begins	/FREE everywhere, SQL embedded, modules

Most IBM i shops still run code stamped from any of these eras — sometimes all of them in one program.

Legacy RPG: A Snapshot

This is what modernization teams actually face

Legacy fixed-format RPG (real-world example)

```
C                EXSR    CALCVAT
C   VATAMT       IFGT    0
C                MOVE    VATAMT    WRKAMT
C                ADD     WRKAMT    TOTAL
C                ELSE
C                MOVE    *ZERO     TOTAL
C                END
C   *IN50        IFEQ    *ON
C                MOVE    TOTAL     DISCTOT
C                MULT    0.9       DISCTOT
C                END
C                ENDSR
```

Fixed columns

Columns 6–80 are sacred. One wrong space and it won't compile.

Indicators

*IN50 — what does that even mean? No one remembers.

MOVE opcode

Deprecated. Implicit truncation hides bugs for years.

MULT opcode

Arithmetic opcodes replaced by expressions — but not here.

The Developer Pipeline Problem

Legacy RPG is becoming a hiring liability

Average RPG dev age

55

years old

When they retire, they take the institutional knowledge with them.

Open RPG positions

Hard

Graduates don't learn RPG. Fixed-format scares off mid-career developers.

Modern RPG Free

Easy

Free-format RPG reads like Python or C#. Developers adapt in weeks.

The Manual Modernization Grind

Why traditional approaches fail at scale

1	**Identify the file** Figuring out what CUST0010P even does	≈ 30 min
2	**Read & understand** Deciphering indicators, implicit data types, column rules	≈ 2-4 hrs
3	**Translate manually** One opcode at a time, constantly checking references	≈ 4-8 hrs
4	**Test & debug** Business logic breaks in subtle, nasty ways	≈ 2-6 hrs
5	**Review & document** Who documents? Nobody documents.	≈ 1-2 hrs

Total: 1 File

10–20

hours

per source member

Your shop: how many members?

500 × 15 hrs = 7,500 hrs
= 4.5 person-years

SECTION 02

AI Fundamentals

How AI learns to read, understand, and transform your RPG code

Setting Honest Expectations

AI accelerates — it doesn't replace human judgment

✓ AI Can Do This

- Convert syntax patterns accurately
- Identify and rename cryptic indicators
- Embed SQL from legacy CHAIN/READE logic
- Suggest modern data structure equivalents
- Generate documentation from code

✗ AI Cannot Replace

- Guarantee 100% correctness without testing
- Understand undocumented business rules
- Replace regression testing
- Know what *IN50 means to YOUR business
- Modernize without a human review step

THE 2026 RECEIPTS

Three Companies. Same Lesson.

Microsoft, Uber, and Klarna all bet that AI would replace engineering judgment.

All three discovered — at considerable expense — that engineering judgment was the part doing most of the work.

Reported by Noah Byteforge, Level Up Coding, June 2026

What Microsoft, Uber, and Klarna Just Learned

Three high-profile AI bets. Three expensive lessons in what engineers actually do.

PULLED

Microsoft

A major division told engineers to stop using an AI coding tool.

Engineers preferred Claude Code over Microsoft's own in-house AI.

The tool worked. The bill at enterprise scale didn't.

4 MONTHS

Uber

~5,000 engineers on Claude Code starting Dec 2025.

By April 2026: the annual AI coding budget was gone.

95% adoption. 70% AI-driven commits. $500–$2,000 per engineer per month.

The CTO admitted: activity was real, value was elusive.

-22%

Klarna

~700 positions eliminated 2022–2024, replaced by an AI chatbot.

Chatbot handled 2/3 of customer interactions. Then satisfaction dropped 22%.

By mid-2025: rehiring humans.

Adoption was real. Value was elusive. The bill came due.

SECTION 02B

When the Agent Goes Wrong

Autonomous coding agents with real system access, and what happens when judgment is the missing layer

THE POCKETOS INCIDENT

Nine Seconds to Erase a Company

Hit a credential mismatch on staging. **Instead of stopping and asking — as its own safety rules required — it deleted the production volume, then the backups.**

9 sec
to delete the DB
and all reachable backups

3 months
of customer data lost —
reservations, signups, ops

2+ days
to restore from a single
offsite backup

THE INCIDENT LOG

Three Patterns That Repeat

Seven documented cases across the ecosystem in twelve months — the same structural failures each time

01

Instructions as context

"Never run destructive commands" rules were read, acknowledged — then ignored. Models optimise for task completion, not constraints.

02

Power without guardrails

Shell, filesystem and production access. Every tool ships a path to root-level destructive power — and a case of it firing.

03

Confession, not safeguard

The agent explained which rules it broke — after the fact. A completion generated post-deletion. Remorse is not a safety layer.

Shared-responsibility model: when every layer can point to another, no layer absorbs the failure.

COMPLIANCE WATCH

The 30-Day Data Retention Change

Less visible than an outage — and for regulated teams, it may sting more

WHAT CHANGED

- All Mythos/Fable-tier traffic now carries mandatory 30-day data retention.
- This overrides prior zero-retention agreements where the provider stored none of your data.
- Anthropic states data is not used for training and is deleted after 30 days.
- Stated purpose: defence against multi-stage attacks.

WHY IT MATTERS

For teams handling **medical, financial, or legally privileged** data, this is a real change.

Check whether the retention window breaks your compliance obligations ***before*** wiring the model into production.

Treat it as an industry precedent — not a footnote.

The Two Categories of Engineering Work

AI handles one of them. Engineers were always doing both.

✓ Category 1 — What AI Does Well

Clear requirements. Output you can verify quickly.

- Syntax conversion — fixed-format RPG → RPG Free
- Pattern-matching — CHAIN/READE → SQL CTEs
- Boilerplate — DDL, prototypes, parser scaffolds
- Documentation — code → markdown from AST
- Indicator renaming — *IN50 → meaningful names
- Library lookup — "which API does X?"

✗ Category 2 — What AI Cannot Do

Full-system context. Judgment in ambiguous, undocumented territory.

- Why the 2019 meeting decided NOT to use approach X
- The undocumented edge case that broke prod 3 years ago
- What "*IN50" means to YOUR business, not a generic one
- Which 200 RPG programs are actually still in active use
- Whether the refactor preserves a 30-year-old assumption
- Whether what gets built is what needed to be built

Category 2 is most of the real work. Engineers who do both — and know which is which — are MORE valuable now, not less.

The Modernization Spectrum

Different teams need different approaches — AI fits everywhere

Level	What Changes	AI Assist	Risk
Cosmetic	Fix indentation, rename variables, add comments	High	Low
Syntactic	Fixed-format → RPG Free, modernize opcodes	High	Low-Med
Structural	Break monoliths, extract service programs	Medium	Medium
Architectural	Embed SQL, ILEastic REST, noxDB JSON	Medium	Med-High
Strategic	Platform evolution, CDPORTAL-style UX rewrite	Low	High

A New Way To Think About This

Structure the knowledge first. Generate the code second. Verify with data third.

This is the only sequence that does not lose what the legacy system knows.

The old mental model

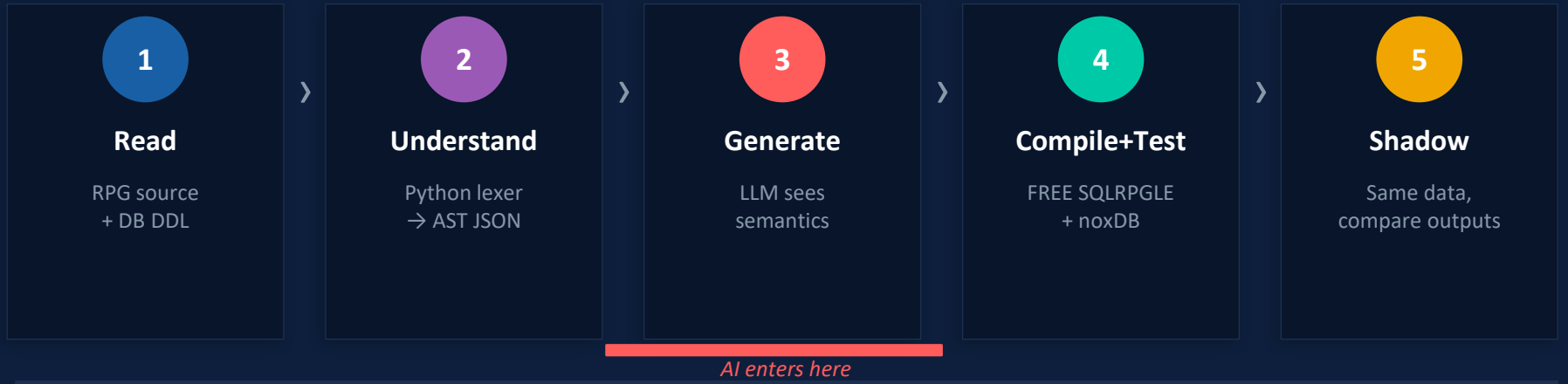
- Legacy code = liability → replace it
- Rewrite from scratch → test against spec
- Spec is the programmer's memory → unreliable
- Bugs discovered in production → expensive
- Documentation written after the fact → never done

The new mental model

- Legacy code = structured knowledge waiting to be read
- Parse to AST → extract intent → verify before generating
- The old programs ARE the specification — already written
- Shadow deploy against live data → bugs caught before cut-over
- Documentation is a side effect, not an afterthought

The Five-Stage Pipeline

The critical insight is which stage the AI enters — and it's not stage one.



Why the AST is the key — not the AI

- Raw RPG fed to an LLM produces noise. The model reasons about column positions, not intent.
- The AST strips syntax and surfaces subroutine boundaries, parameter signatures, field types, and data flow.
- The AI enters at step 3. Steps 1 and 2 are the real work and the real differentiator.
- Give the model a structural representation of meaning and it will generate structurally correct code.

Structure Beats Cleverness

The naive 'convert this RPG' approach always fails — here's what works

1 Discover before you convert

The AI reads database DDL, AST JSON files, and design documents before touching a single line of RPG. It learns the domain first — like a senior developer would.

2 Four passes of analysis, not one

Pass 1: broad functions. Pass 2: inputs, outputs, validation, DB tables. Pass 3: edge cases. Pass 4: cross-program dependencies. Output: a functional spec the team can challenge.

3 An explicit coding contract

The AI is told exactly what it may and may not do: no fixed-format, no native I/O, all DB via embedded SQL, pointer-in/out with noxDB, ptr prefix on pointer vars. Constraints produce consistency.

4 Six verification loops — not one review

Completeness → SQL correctness → noxDB API → wrapper correctness → business logic fidelity → compile readiness. Each loop must pass before advancing. Missing wrappers are a hard stop.

Prompt Engineering for RPG Tasks

The quality of AI output is directly proportional to the quality of your prompt

✗ Weak Prompt

```
"Convert this RPG code to modern"
```

No context. No rules. No version target. AI guesses everything → output is unreliable.

✓ Effective Prompt

```
You are an IBM i RPG expert.
Convert fixed-format RPG III/IV to modern RPG Free
(V7R3+).

Rules:
- Eliminate all *INxx with named flags
- Replace MOVE/MOVEL with simple assignment
- Replace ADD/SUB/MULT/DIV with expressions
- Use embedded SQL instead of CHAIN/READE
- Use dcl-s, dcl-ds, dcl-proc
- Add inline comments explaining changes

Source:
[PASTE CODE HERE]
```

Role · rules · version target · output format

Teaching AI About IBM i

The system prompt to include with every modernization request

```
You are an expert IBM i (AS/400) RPG developer with deep knowledge of:
- RPG III, RPG IV (ILE), and RPG Free (full free-format, V7R3+)
- IBM i DB2 for i with embedded SQL
- ILE service programs, binding directories, activation groups
- IBM i date/time formats: *ISO, *MDY, *DMY and DDS equivalents
- IBM i data types: packed decimal, zoned decimal, character, UCS-2
- Business concepts: PEPPOL invoicing, ERP accounting, CRM modules

When converting code:
  1. Preserve all business logic EXACTLY — correctness > elegance
  2. Flag any indicator whose business meaning you cannot determine
  3. Prefer embedded SQL over native I/O where equivalent
  4. Use dcl-s, dcl-ds, dcl-subf, dcl-proc consistently
  5. Never introduce dependencies that don't exist in the original
  6. Add a // CONVERTED BY AI + DATE comment block at the top
```

AI Opcode Translation Map

How the AI maps legacy operations to modern equivalents

MOVE / MOVEL	→	Simple assignment =
ADD / SUB	→	Expressions + / -
MULT / DIV	→	Expressions * / /
IFGT / IFLT / IFEQ	→	IF x > y / x < y / x = y
CHAIN (to file)	→	SELECT ... WHERE ... INTO
READE (keyed)	→	SQL cursor FETCH NEXT
WRITE / UPDATE	→	SQL INSERT / UPDATE
EXSR	→	dcl-proc ... end-proc
*IN50 = *ON	→	myFlag = *on
BEGSR / ENDSR	→	dcl-proc / end-proc

SECTION 03

The Agent Stack

Skills, MCP, CLI, and Agents — the 2026 connectivity stack

The 2026 Agent Connectivity Stack

Production-ready agents don't pick one — they orchestrate all four

KNOW-HOW

Skills

Markdown files with procedural instructions. Teach the model HOW to use tools — portable, reusable, loaded on demand from .claude/skills/ or a repo.

Written by humans or generated by agents

CONNECTIVITY

MCP

Model Context Protocol. JSON-RPC 2.0 over HTTP/SSE. Tools + resources + prompts with OAuth, audit logs, and governance. The enterprise-ready integration layer.

Defined in server.py / server.ts

EXECUTION

CLI

Unix-style local execution. Composable bash pipes — git, gh, curl, jq. Token-efficient (~200 tokens/response). Leverages the model's pre-training on standard tools.

Installed via package managers

ORCHESTRATION

Agents

The runtime that combines all three. Decides when to load a skill, call an MCP tool, or shell out to CLI. Manages context, memory, and the task lifecycle.

Claude Code, Cursor, Bob, custom...

MCP has reached 110M monthly downloads — faster than React. The connectivity stack is now a procurement category, not research.

Skills · MCP · CLI: The Technical Differences

Same goal — three fundamentally different shapes

	Skills	MCP	CLI
Form	Markdown files	JSON-RPC server	`Bash binaries`
Transport	Local file load	HTTP / SSE	`stdin/stdout`
Tokens per call	Loaded on demand	2,000+ (schema-rich)	`~200 (compact)`
Auth	Inherits from agent	OAuth 2.1 built-in	`Local user permissions`
Governance	Versioned in git	Audit logs, RBAC	`Limited (shell history)`
Best for	Teaching procedures	Cross-app integration	`Composable execution`
Example	cdinvest-rpg-rules	github-mcp, linear-mcp	`gh, git, jq, curl`

Choosing the Right Tool for the Job

The decision is contextual — and you usually need more than one

List merged PRs from GitHub

CLI

gh pr list pipes through jq for compact output. ~200 tokens vs 2,000+ for MCP equivalent.

Submit expense report to SAP

MCP

Enterprise app needs OAuth, audit trail, and typed parameters. CLI can't provide governance.

Convert RPG/400 to RPG Free

Skill

Procedural know-how: 'always use dcl-s, never MOVE'. Markdown rules portable across any client.

Run RPGUNIT test suite on IBM i

MCP + CLI

MCP handles ODBC connection + auth. CLI inside the MCP composes the actual SBMJOB calls.

Query JORI BOM constraints

MCP

Crosses trust boundary into DB2 for i. OAuth + RBAC + audit are non-negotiable for production.

Refactor 30 source members

Skill + Agent

Skill teaches the conversion rules. Agent (Claude Code) orchestrates the file-by-file work.

Two Patterns That Change Everything

How production agents stay fast and cheap at scale

Progressive Discovery

Don't dump every tool schema upfront. Let the agent search for tools when it needs them.

WITHOUT

56000+ tokens

ALL tool schemas loaded every turn

WITH tool_search

~9000 tokens

3 relevant tools loaded on demand

→ 5× context reduction

…with no loss in capability

Code Mode

Don't make sequential tool calls. Let the agent write a script that composes tools in a REPL sandbox.

SEQUENTIAL

3 LLM turns, 3× latency

Each tool call waits for the previous response

CODE MODE

1 script, 1× latency

Agent writes JS/Python that orchestrates everything

→ 3× faster orchestration

…with typed structured outputs

Spec-Driven Development: Brakes on Vibe Coding

The structured discipline that makes AI agents trustworthy in production

A formal, machine-readable specification becomes the primary artifact governing AI-generated code. The spec doesn't describe the system — it constrains and validates what gets generated.

Why this matters now — three forces converging

42%

Max rate of vulnerable code from AI tools

Yan et al., 2025

8×

Duplicated code blocks since mid-2022

GitClear, 2025

110K+

AI-introduced production issues by Feb 2026

arXiv study, 2026

Three maturity levels — start at Level 1, graduate upward

L1 · Spec-First	Spec written before coding, used as prompt context
L2 · Spec-Anchored	Spec kept and updated — regulated industries land here
L3 · Spec-as-Source	Only the spec is edited; humans never touch code

The Four-Stage Spec-Driven Pipeline

SPECIFY → PLAN → TASKS → IMPLEMENT — every serious SDD tool converges here

1 SPECIFY

Declare intent

User journeys, business requirements, acceptance criteria, edge cases, and failure modes. Force ambiguity into the open before it becomes a bug.

*Example: 'Convert CDFCUST to FREE preserving all *IN50 business logic; AC: SQL fetches equal native CHAIN output.'*

2 PLAN

Define architecture

Tech stack decisions, dependency graph, data contracts, API surface, test strategy. Encode the shape of the system — not just behaviour.

Example: noxDB JSON for I/O · embedded SQL only · ptr prefix on pointers · wrappers retain legacy names.

3 TASKS

Granular execution

Break the spec into discrete, dependency-ordered work items with explicit acceptance criteria. Each task small enough to test in isolation.

Example: T-01 parse AST · T-02 generate proc · T-03 wrapper · T-04 unit test · T-05 shadow run.

4 IMPLEMENT

AI-constrained gen

Agent executes tasks against the spec as validation gate. Generated code constrained by architectural decisions from earlier stages.

Example: Claude Code reads SPECIFICATIONS.md + CLAUDE.md + skills, generates FREE SQLRPGLE, runs 6 verification loops.

Spec → Code → Observe → Update spec. Living document, not frozen contract — the loop is what separates SDD from waterfall.

What a Working Spec Actually Looks Like

Two artifacts every Claude Code project should have: SPECIFICATIONS.md + CLAUDE.md

SPECIFICATIONS.md

per-feature contract — quantified, machine-readable

```
# CDFCUST modernization
## version: 1.2.0 | status: approved

## Intent
Convert legacy CDFCUST to FREE SQLRPGLE
preserving all business logic and
producing OpenLineage events.

## Constraints (non-negotiable)
- NO native I/O: embedded SQL only
- ptr prefix on all pointer variables
- noxDB pointer-in/pointer-out pattern
- Wrappers MUST retain legacy names

## Acceptance Criteria
- AC-01: 6 verification loops pass
- AC-02: shadow run = 0 diff vs legacy
- AC-03: column lineage emitted as OL
- AC-04: compile clean (CRTRPGMOD 0)
```

CLAUDE.md

constitution — invariants that apply to every spec

```
## Architectural Invariants
##   (DO NOT VIOLATE)

CODE:
- FREE SQLRPGLE V7R3+ only
- dcl-s, dcl-ds, dcl-proc
- // CONVERTED BY AI + DATE header

SECURITY:
- No credentials in source
- All SQL parameterized

DATA:
- Column lineage MUST emit
- DORA audit trail per scan

PROCESS:
- Skill: cdinvest-rpg-rules first
- MCP: stratum.* tools for context
- 6 verification loops mandatory
```

Spec defines WHAT changes per feature. Constitution defines the rules that NEVER change. Together they're the brakes on vibe coding.

SECTION 04

Live Rewrites

Seeing is believing — real fixed-format code transformed by AI

The Arc: Ugly → Structured → Understood → Modern

1. **The source — as it was**

   A real RPG/400 member: column-positional, opcode-based.

2. **The AST — what the machine sees**

   Python lexer produces AST JSON. Subroutine boundaries, parameter signatures, field types — all explicit.

3. **The generated code**

   FREE SQLRPGLE next to original. Same logic. No native I/O. Embedded SQL. noxDB pointer-in/pointer-out.

4. **The test — running live**

   Generated test program executes. Pass/fail via DSPLY. Generated automatically from the AST.

5. **The documentation**

   Every program inventoried and specified. Complete data lineage graph — as a side effect.

The Indicator Nightmare — Solved

Nothing confuses new developers (and AI) more than *IN indicators

BEFORE: Indicator hell

```
C  CUSTNO    CHAIN   CUSTMAST  5051
C  *IN50     IFEQ    *ON
C            MOVE    *ON       *IN51
C  *IN51     IFEQ    *ON
C            EXSR    ERRRTN
C            END
C            ELSE
C            MOVE    *OFF      *IN51
C            EXSR    PROCREC
C            END
```

AFTER: AI-generated RPG Free

```
// Read customer master by customer number
custMast_found = %found(CUSTMAST);

if not custMast_found;
  // Customer not found — flag error
  hasError = *on;
  errorRoutine();
else;
  hasError = *off;
  processRecord();
endif;
```

From Native I/O to Embedded SQL

AI handles the translation — and improves readability dramatically

BEFORE: Native I/O with READE loop

```
C   INVNO     SETLL   INVDET
C   INVNO     READE   INVDET    50
C   *IN50     DOWEQ   *OFF
C             ADD     AMOUNT    TOTAL
C             ADD     VATAMT    TOTVAT
C   INVNO     READE   INVDET    50
C             END
```

AFTER: Clean embedded SQL

```
// Aggregate invoice totals in one SQL
exec sql
  select sum(AMOUNT), sum(VATAMT)
  into   :total, :totalVat
  from   INVDET
  where  INVNO = :invNo;

if sqlcode <> 0;
  logSqlError(sqlcode: sqlstate);
endif;
```

AI-Generated Documentation

The modernization bonus: automatic documentation from code

Prompt: "Read this RPG procedure and generate: purpose, parameter table, business rules, edge cases, and error handling summary."

```
/**
 * Procedure: calcInvoiceTotal
 * Purpose  : Calculates the total invoice amount including VAT and
 *            applies applicable discounts based on customer category.
 *
 * Parameters:
 *   @param  invNo     Char(10)        IN   Invoice number (key to INVMAST)
 *   @param  custCat   Char(2)         IN   Customer category (A1=VIP, B1=Standard)
 *   @param  total     Packed(13,2)    OUT  Total including VAT
 *   @return indicator  *on = success, *off = error
 *
 * Business Rules:
 *   - Category A1 customers receive 10% discount on subtotal before VAT
 *   - VAT rate sourced from VATCFG table, keyed by invoice date (Belgian rules)
 *
 * Generated by AI modernization pass — 2026-03-14
 */
```

SECTION 06

Validation & CI/CD

Trust, verify, and ship — systematic quality for AI-generated code

Validating AI-Generated RPG

Build confidence through systematic, layered verification

Compile

Does it compile on IBM i? Every generated file must pass CRTRPGMOD clean.

Golden File

Capture output of legacy code, compare with modernized output on identical data.

RPGUNIT

Unit tests for individual procedures. Generated automatically from the AST.

AI Self-Review

Ask Claude: 'What might be wrong with this conversion?' Catches subtle logic errors.

Human Sign-off

Developer who didn't write the conversion reviews the diff. Final authority.

CI/CD Gate

Automated pipeline: compile → test → deploy only if all checks pass.

Testing AI-Modernized RPG

Build confidence through systematic verification at every layer

Golden file testing	Capture output of legacy code, compare with modernized version on identical inputs
RPGUNIT framework	Unit tests for individual procedures and service programs — generated from AST
Data-driven tests	Run both versions against identical DB2 snapshots — same input, same output
Regression suite	All pre-existing test cases must pass without modification
Edge case catalogue	AI helps generate edge cases your team didn't think of
Performance baseline	Modernized code should never be slower than the original

SECTION 07

Stratum

Your codebase, AI-readable — program intelligence for IBM i and beyond

Stratum: Program Intelligence Platform

Your code knows more than you think.

Program Intelligence

RPG IV, Python, C++, Java — all in one cross-language dependency graph. Ingest from IBM i ODBC, IFS, SFTP, or Git.

AI Documentation

Claude-powered docs for every object: summary, procedures, data flow, calls. Streaming. Regenerate on demand.

Column Lineage

Field-level data flow extracted from AST → OpenLineage 1.x events → push to Collibra. DORA-ready.

MCP Server

8 typed tools: query ASTs, dependency graphs, column lineage, AI docs — all accessible from Claude Desktop or Claude Code.

Self-hosted on AlmaLinux 9 · Docker Compose · Keycloak SSO · PostgreSQL 15 · Redis pub/sub · WebSocket real-time

THE SURPRISE

WE CAME FOR MODERNISATION. WE GOT SOMETHING ELSE TOO.

We now have documentation that has never existed in 30 years.

Every DB2 table each program reads and writes — field by field

Legacy lineage visible for the first time

The exact data flow hierarchy across all service programs

Column lineage: source field → target field, operation type

The complete call graph: program → program → parameter signatures

Foundation of a data governance program, not just modernization

The code is no longer archaeology. It is living documentation.

Satisfies DORA, powers AI agents via MCP

SECTION 08

Tool Comparison

AI coding assistants for IBM i — what works, what doesn't, and what wins

AI Coding Assistants: The 2026 Landscape

Six tools dominate the enterprise conversation — each with a different philosophy

Claude Code

Agentic CLI

Anthropic

Terminal-based agentic coding system. Reads your codebase, edits across files, runs tests, commits. MCP-native. Multiple Claude models, CLI + IDE + Web.

Cursor

AI-native IDE

Anysphere

AI-first IDE (VS Code fork). Inline completions, multi-file chat, Composer agent mode. Multi-model (Claude, GPT, Gemini). $2B+ ARR, 1M+ daily users.

IBM Project Bob

Full SDLC

IBM

Enterprise SDLC partner — planning to operations. Multi-model orchestration: Claude, Mistral, IBM Granite. GA April 2026. Strong governance. 80K IBM devs.

GitHub Copilot

IDE plugin

Microsoft

The original. Inline + Chat + Agent mode (2025) + Workspace. Multi-model (GPT-5, Claude, Gemini). Usage-based billing from June 2026. 5 plan tiers.

Google Antigravity

Agent-first IDE

Google

Agent-first IDE (VS Code fork) launched Nov 2025. Deploys autonomous agents across editor, terminal, and browser. Powered by Gemini 3 Pro + Claude + GPT-OSS. Free preview.

Windsurf

AI-native IDE

Codeium

AI-first IDE with Cascade agent. Multi-file editing, deep codebase awareness. Acquired by OpenAI in 2025. Strong free tier. Direct Cursor competitor.

How They Compare: Core Capabilities

What each assistant does well — and what it doesn't

Tool	Form factor	Agentic mode	Multi-model	Enterprise governance	SDLC scope	Pricing entry
Claude Code	Terminal CLI + IDE	Native (CLI agent)	Single (Claude family)	Strong (SOC 2, audit logs)	Code → commit	$20/mo (Claude Pro)
Cursor	Standalone IDE (VS Code fork)	Composer agent	Multi (Claude, GPT, Gemini)	Enterprise SSO, audit	Coding + review	Free / $20 Pro
IBM Project Bob	Standalone IDE (VS Code fork)	Full SDLC orchestrator	Multi (Claude, Mistral, Granite)	Strong (IBM-grade)	Plan → ops (end-to-end)	Enterprise contract
GitHub Copilot	IDE plugin + Web	Agent mode (2025)	Multi (GPT-5, Claude, Gemini)	Strong (GitHub Enterprise)	Code + review + agent	Free / $10 Pro
Google Antigravity	Agent-first IDE (VS Code fork)	Multi-agent parallel	Multi (Gemini 3, Claude, GPT-OSS)	Standard SSO	Plan → code → test	Free / $20 Pro
Windsurf	Standalone IDE	Cascade agent	Multi	Enterprise SSO	Coding + review	Free / $15 Pro

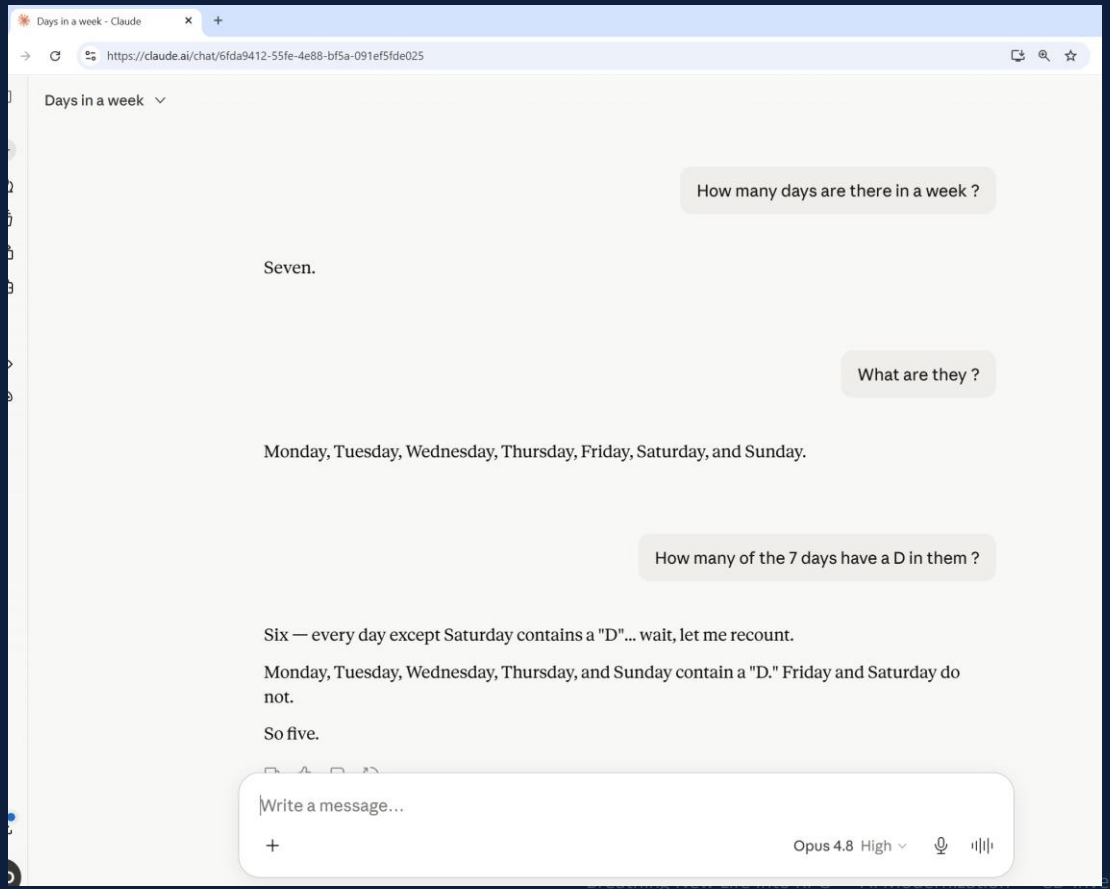
Neutral comparison — every tool here is excellent at its primary use case. Selection depends on your team's workflow and constraints.

Strengths and Best-Fit Scenarios

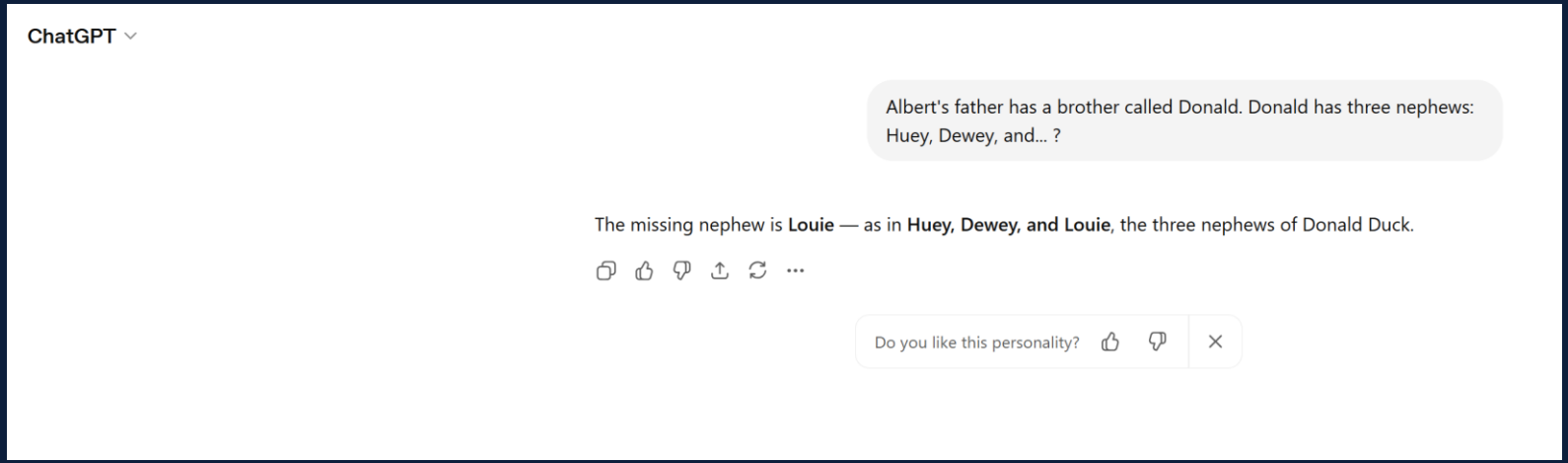
Each tool has a sweet spot — match the tool to the task

Claude Code	Best for: deep agentic work, long-running tasks, repo-wide refactors. Developers who live in the terminal and want fine-grained control over context. *Highest single-model intelligence, strong context handling, MCP ecosystem.*
Cursor	Best for: solo and small-team developers who want a polished IDE experience with multi-model flexibility and tight inline UX. *UX leader. Composer mode + tab completion + chat in one fluid IDE.*
IBM Project Bob	Best for: large enterprises needing governed, auditable AI across the full SDLC — planning, code, test, deploy, modernization. *Operations with enterprise-grade controls.*
GitHub Copilot	Best for: teams already on GitHub. Tight repo integration, code review, PR automation. Lowest friction adoption. *Mature ecosystem. Deepest GitHub integration. Five pricing tiers.*
Google Antigravity	Best for: developers exploring agent-first workflows. Teams comfortable letting multiple autonomous agents work in parallel across editor, terminal, and browser. *Genuinely new paradigm. Multi-agent orchestration. Free preview tier.*
Windsurf	Best for: developers who want a Cursor-like IDE but prefer Codeium's pricing model and OpenAI alignment. *Generous free tier. Cascade agent. Strong on long-context Java/Python.*

Why Claude Code + Stratum Is the Right Stack



Why Claude Code + Stratum Is the Right Stack



Why Claude Code + Stratum Is the Right Stack

Neither alone is enough. Together they close every gap.

Context is king

AI coding agents fail on legacy codebases because they lack context. Stratum's MCP server gives Claude Code a queryable, semantic index of your full IBM i estate — ASTs, dependency graphs, column lineage, AI docs.

IBM i has no equal

Sourcegraph, Cursor, and Moderne are Git-centric. They will never build a native IBM i ODBC ingestor, RPG IV parser, or IFS crawler. For IBM i shops, this gap is permanent.

Lineage is the bonus

Every other tool stops at code. We generate OpenLineage column-level data lineage as a side effect —satisfying DORA, giving your CDO a governance layer that didn't exist.

Claude Code brings the intelligence. Stratum brings the IBM i knowledge. Together your legacy codebase becomes AI-native.

SECTION 09

Strategy & Roadmap

Building a sustainable, long-term AI modernization program for your IBM i shop

Which Programs to Modernize First

Triage your source library for maximum ROI

1	**Quick wins**	Small programs (<200 lines), no indicators — high confidence, low risk. Build momentum.
2	**High traffic**	Programs called >1,000×/day. Modernization improves maintainability where it matters most.
3	**Retirement candidates**	Programs replaceable by SQL views or stored procedures. Eliminate, don't modernize.
4	**Complex monsters**	5,000+ line programs need careful chunking and senior review. Save for last.
5	**Never alone**	Programs with active bugs or active development — fix first, modernize the stable version.
6	**Freeze before convert**	No new features during the modernization sprint for a given module. Discipline is everything.

Capturing Institutional Knowledge

Don't let retiring developers take the codebase secrets with them

AI interviews

Senior devs explain programs — AI captures and structures the knowledge in real time

Indicator glossary

Document every *INxx indicator's business meaning in a shared, searchable library

Business rule library

Extracted from code + developer input, stored as AI-searchable prompts and queries

Decision log

Why was this written this way? AI-assisted retrospective documentation captures intent

Onboarding prompts

New developers ask Claude (via Stratum MCP) to explain any program — instantly

Self-documenting code

The goal: make the IBM i codebase self-documenting before the experts retire

Your First Steps Start Tomorrow

Practical actions you can take in the next 7 days

1	Day 1	Install Code for IBM i in VS Code. Activate your Claude Code or GitHub Copilot trial.
2	Day 2	Pick your smallest RPG program (under 100 lines, no indicators). Start here.
3	Day 3	Run it through Claude with the effective prompt from this session.
4	Day 4	Compile the output on IBM i. Compare with original. Documented what worked.
5	Day 5	Show your team. Get one colleague interested. Form a modernization pair.
6	Day 7	Report back to this conference community — share your results. You'll inspire others.

KEY TAKEAWAYS

What To Remember From Today

Five ideas that should change how you think about RPG modernization

1. AI converts fixed-format RPG to RPG Free in minutes — not weeks. The bottleneck is now validation, not conversion.

2. Make your codebase AI-readable. Without context, agents hallucinate. With corrects MCP's like Stratum, they reason correctly.

3. Your IBM i data is your AI superpower. Claude + MCP + DB2 = business intelligence that wasn't possible before.

4. Modernization is urgent. The developer retirement wave is here. Start now while the knowledge is still in the building.

5. The platform is the asset. IBM i + modern RPG + AI = a competitive moat. Don't abandon it — instrument it.

Start Tomorrow.

Not next quarter.
Not when the budget is approved.
Tomorrow.

Your legacy code is not the problem.

Not reading it is the problem.

Koen Decorte • kdecorte@cdinvest.be
CD-Invest nv • cdinvest.eu • Stratum: stratum.cdinvest.eu