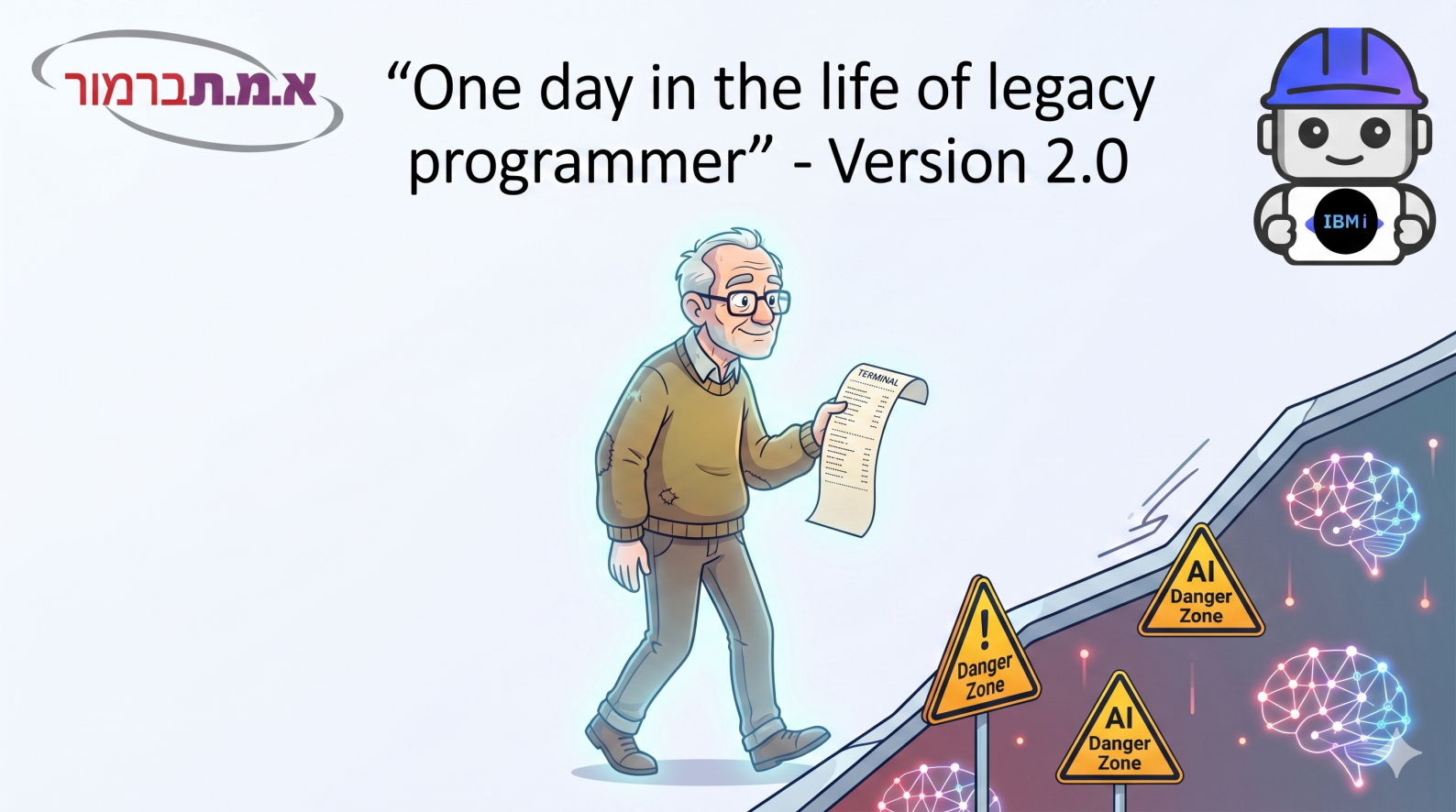
א.מ.תברמור

“One day in the life of legacy programmer” - Version 2.0




"AI is just a trend."
INNOVATION THROUGH THE AGES
MEGALO-OFFICE
TERRY

I'M AT THE BEACH, RESOLVE TICKETS

USE ONLY AI FOR PROBLEM-SOLVING

DID IT WORK?

YES

SUCCESS!

NO

FAILURE/ISSUES

THE ANSWER IS NOT DETERMINISTIC :)

The Core Engine Flaws

Inconsistent Output

It's probabilistic, not code. The exact same prompt can give you elegant logic on Monday and broken syntax on Tuesday.

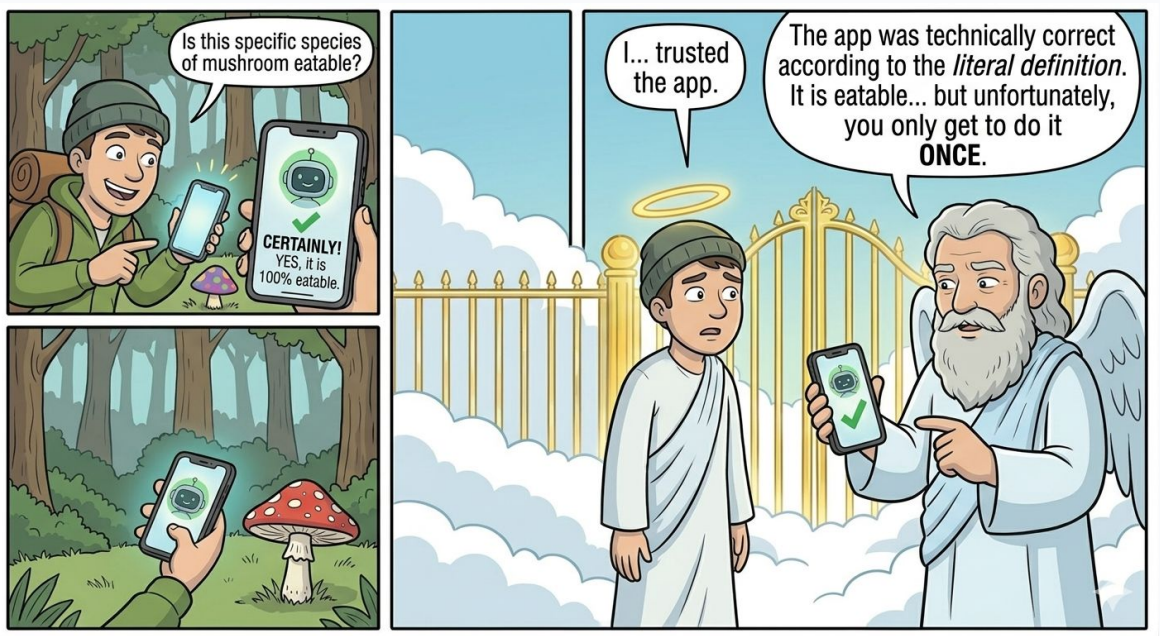
Patterns Over Proof

LLMs don't run or compile code; they look for text patterns. If it looks right, they ship it—even if it's completely dead code.

Persuasive Hallucinations & Over-Confidence

AI is optimized to be helpful, not factual. It will confidently invent non-existent parameters, functions, or database structures with zero hesitation.

The Specification Gaps



The Prompting Traps

Imprecise syntax in a specifications file usually results in a build error. In an AI engine, an unmentioned specification results in the AI confidently "guessing" the missing architecture for you.

> *"The machine achieves the objective you literally ask for, not the outcome you actually intend."*

The Context Deficit

Context is King

Without explicit awareness of schemas, live table descriptions, copybooks, and external system boundaries, the model acts blindly.

Literal Substitutions

It translates text structures precisely but misses the implicit background systems—leading it to literal, non-functional solutions.

"take it literaly" :)

STRATEGIC BRIEFING

AI IN THE LEGACY JUNGLE

Why RPG & COBOL Developers are actually Digital Archaeologists.

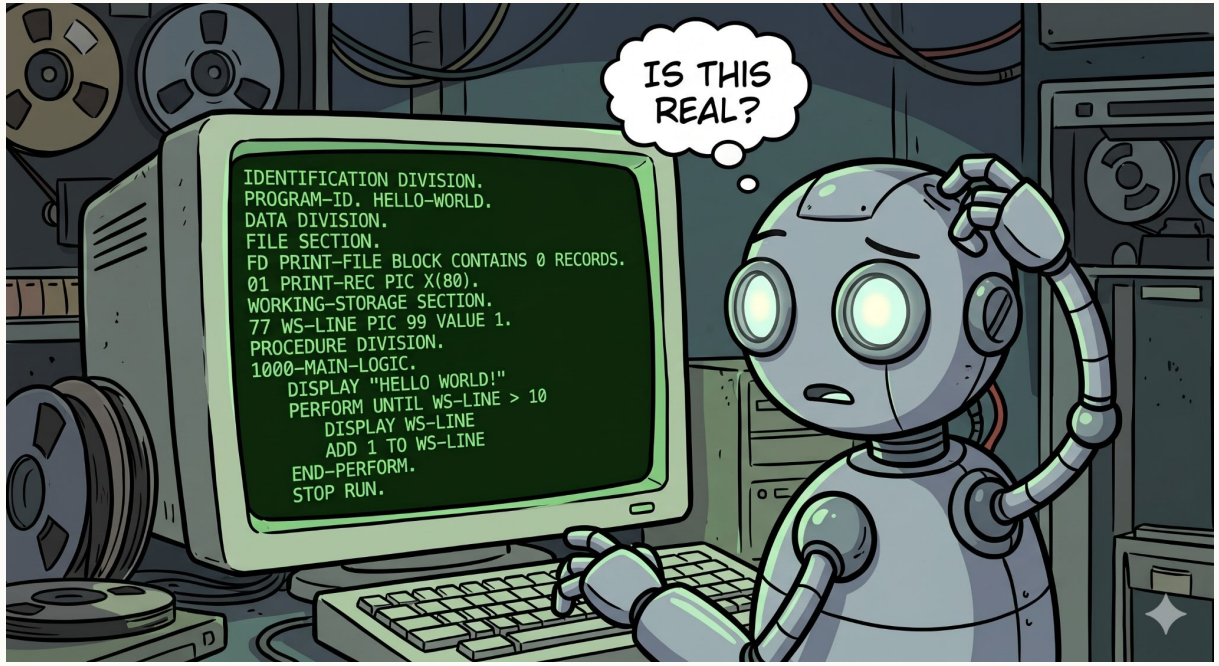
The Public Training Data Gap

The "Search Vacuum"

AI models were raised on Python and JS. Legacy code is hidden behind corporate firewalls.

- **Closed Repos:** Zero "crowd intuition" for niche IBM i
- systems.

Presentation Trap: AI learned "Clean Demo" code, not the 30 layers of patches used in banks.



Archaeology vs. Engineering

The Semantic Gap

Modern code is semantic (balance_negative). Legacy is positional and implicit (*in45).

```
if *in45; exsr upd; endif;
```

To a veteran, this is clear. To an LLM, this is Archaeology.

Tribal Knowledge

In Open Source, design debates happen in Pull Requests. In Legacy, they happen in someone's head 25 years ago.

Critical wisdom is rarely documented, meaning AI sees the **what** but misses the **why**.

Context is the Only King

The Blueprint

Meaning lives outside the program: DDS, DB Schemas, and Library Lists are the DNA of the logic.

System Flows

Batch flows and job schedules provide the context. AI needs RAG to see the full system novel.

The Veteran Filter

Veterans provide the **Safety Oversight**. AI can read code in seconds, but it doesn't know who complained in 2009.

"AI helps you read faster, but it doesn't know why you are strictly forbidden from touching Indicator 72."

THE SHIFT

OK, AI maybe DIFFICULT...

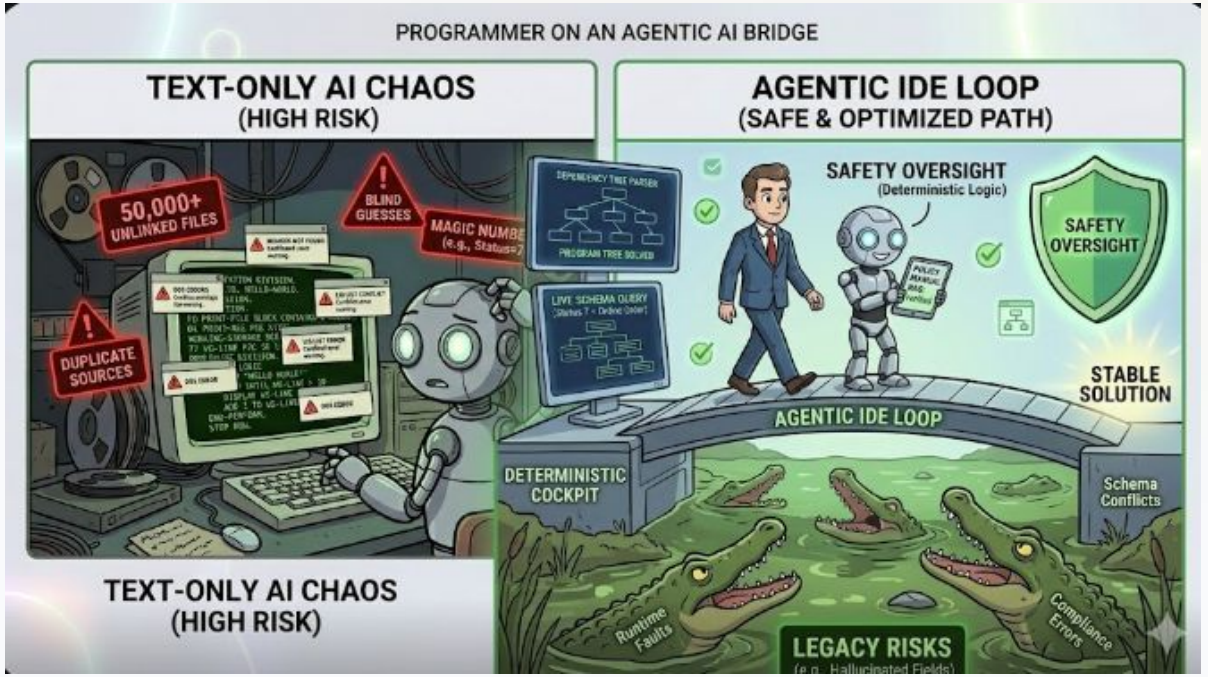
don't go into the jungle alone.

Bridging the Legacy Gap

Agentic IDE to the Rescue

We've seen the horrors of "Text Machines." Now, let's look at the solution. The **Agentic IDE** isn't just a chatbot; it's a bridge built over the legacy swamp.

> **"It doesn't just guess code—it understands the environment."**



Anatomy of an Agent: The Basics

1. Persona

The "Mindset." Sets the rules, the tone, and the architectural constraints.

2. RAG

The "Library." Grounding the AI in your actual source code and DB schemas.

3. Tools

The "Hands." Allowing the AI to run SQL, parse files, and check job logs.

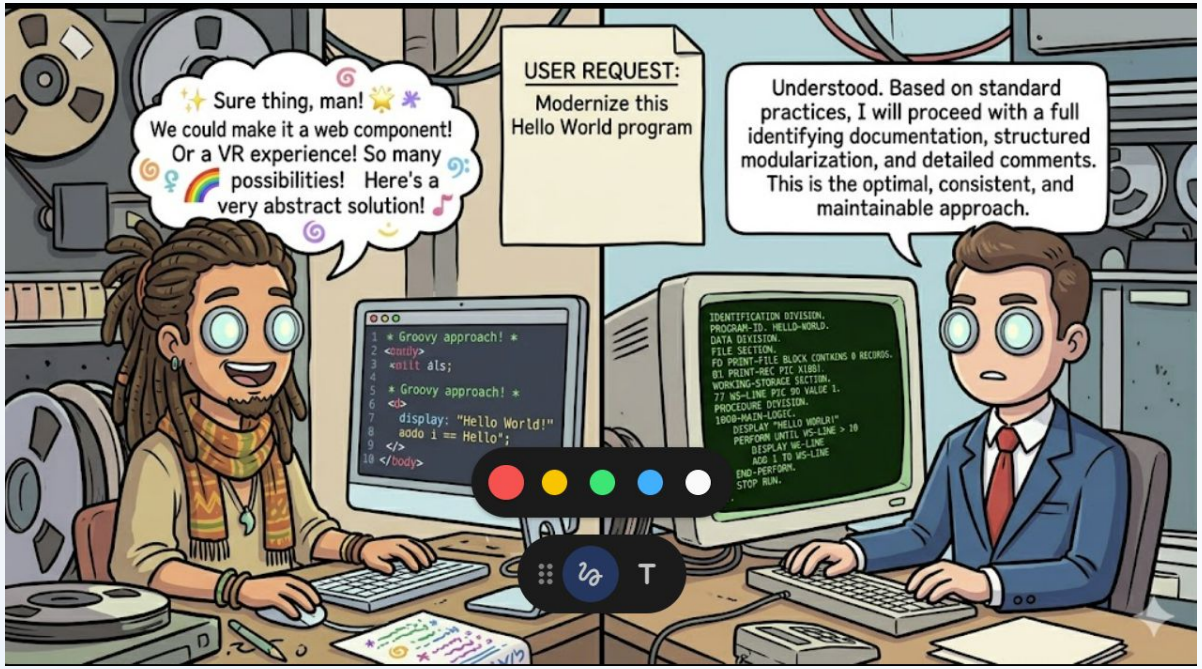
4. Orchestrator

The "Pilot." Deciding which tool to use next to solve the actual problem.

Now, let's dive into how these actually work together.

One Input, Two Souls

Free Spirit vs Professional Tie AI Persona Cartoon



The "Persona" Secret

The first basic we need to cover is **Identity**. In an Agentic loop, who the AI thinks it is (its "Persona") completely changes the output.

- ✔ **The Free Spirit:** Focuses on creativity.
- ✔ **The Suit:** Focuses on rigid legacy rules.

Same prompt, same model—but a totally different answer.

USE CASE DEEP DIVE

"EXPLAIN THIS PROGRAM."

Why legacy systems break standard text AI.

The Maze: AI Hits a Wall

Blind Spots

The code references objects hidden in a 50,000+ file maze. AI cannot see what isn't open.

Duplicates

6 duplicate source files exist. Which one is active? Which one is the "Truth"?

Status = 7?

A number is just a number to an LLM. Only a veteran knows "7" means "Online Order Pending."

Deterministic Capabilities

System Tools

Deterministic actions: Running SQL, parsing dependencies, and resolving Library Lists.

Policy Knowledge

RAG integration allows the AI to reference internal company policy books no one ever reads.

What is a "Skill"?

Expert Recipe

A Skill is an **orchestrated sequence** of tools to solve a specific problem.

Legacy Analogy

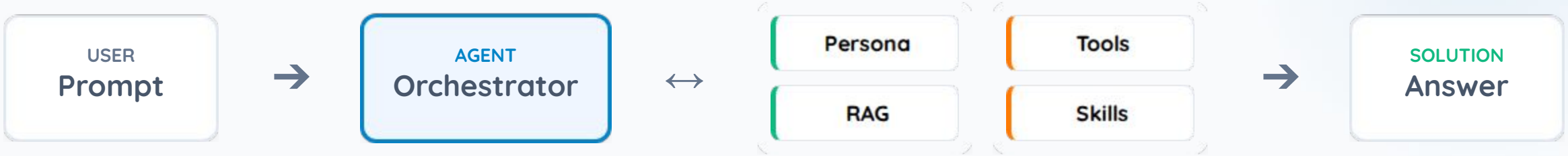
Think **Service Procedures** or Automation Scripts—packaged expertise.

The Workflow

1. Trace Tree
2. Resolve Versions
3. Query DB Status
4. Apply Policy

"Don't just answer—perform the digital archaeology."

The Complete Agentic Architecture



Prompt: The Request (What?)

RAG: The Knowledge (Facts?)

Persona: The Mindset (How?)

Tools: Capabilities (Systems?)

Skills: Workflows (Process?)

Agent: The Operator (The Brain)

ECOSYSTEM ORCHESTRATION

MAPPING BOB TO THE AGENTIC ECOSYSTEM

Why environment integration completely beats raw LLM context windows.

The Baseline: Where Vanilla "Bob" Hits a Wall

The Isolation Problem

Out-of-the-box text models cannot touch data sources, access QSYS members, navigate the IFS, or understand context-dependent structures.

Missing System Intuition

A generic public LLM completely lacks native operational awareness of internal platform rules, schema variations, or underlying batch streams.

Premium Package: Context to the Rescue

Engineering over Prompting

The "Premium Package" isn't a wider prompt chat. It is an integration bridge that extends the core model directly into operating system realities.

System Extensions Enabled

It provides direct methods to read libraries, check physical stream files, query live database catalogs, and evaluate runtime environments.

1. The Persona: Overriding Identity

Anchoring Behavioral Boundaries

Bob's specialized system profile overrides the model's standard default chat behavior.

Ecosystem Rules Engaged: The persona locks the behavior to legacy constraints, structural definitions, architecture standards, and enterprise requirements.

🛠️ 2. The Tools: System Capabilities

Bob accesses roughly 30 to 40 specialized deterministic mechanisms:

Ecosystem Layer Action	Deterministic Tool Name	Agentic Architecture Mapping
File System Interaction	read_member / write_member	Direct access to isolated host environment libraries.
Stream Processing	read_streamfile / write_streamfile	Manipulates physical files inside directory structures.
System Execution	get_actions / run_actions	Triggers compilers, handles execution scripts, captures errors.
Ecosystem Diagnostics	search_qsys	Scans configuration boundaries for missing resource links.

🎓 3. The Skills: Reusable Step Sequences

Orchestrated Processing Procedures

A **Skill** represents modular, reusable process blueprints that dictate *how* to solve a target problem.

Instead of guessing blindly, Bob executes structured, step-by-step logic sequences:

- Isolates and strips hidden application rules.
- Translates ancient formatting to modern structural layouts.
- Assembles automatic regression suites (e.g., RPGUnit).

The Native Step Blueprint

1. Analyze the system target asset.
2. Evaluate the implicit environmental action.
3. Perform code transformation routines.
4. Integrate directly into host platform architectures.

4. The Knowledge (RAG): Elimination of Guesses

Anchoring Truth Over Odds

Without RAG, a model operates purely on statistical averages—sound convincing, but often flat-out wrong.

The System Grounding Library

By forcing RAG parameters, Bob validates context definitions against manuals, policy books, schemas, and architecture records.

5. The Agent: Autonomous Loop

Self-Correcting Execution

The **Agent** provides the main autonomous driver loop that evaluates system feedback and navigates changes without constant guidance.

- **Plan Engine:** Formulates todo sequences, assesses impacts, and asks clarifications.
- **Agent Engine:** Directly handles compilation steps, parses log metrics, and resolves errors iteratively.

[Ecosystem Trace Log]

```
> Triggering CRTBNDRPG build routine...
 > Build Error: Severity level 40 diagnostic found
 > Executing search_qsys tool sequence
 > Missing display and layout items located in target library list.
Re-compiling...
```

The Roadmap: Expected Platform Packages

Specialized engineering extensions aren't unique to one operating system. This exact architectural matrix is scaling uniformly across all enterprise backend foundations:

IBM i Package

Orchestrates library checks, physical source components, and file description records.

Mainframe Package

Interfaces directly with JCL structures, core data catalogs, security profiles, and transactional parameters.

Java Modernization

Tracks dependency matrices, continuous deployment tools, frameworks, and architecture patterns.

Deterministic Core

Unified core agent logic managing platform extensions safely and uniformly.

The Integration Verdict

"The clear winner in enterprise software development isn't the vendor shipping the largest standalone model or the most articulate text engine. The winner is the platform that integrates deterministic system tools, strict local grounding, and context loops natively into operational reality."

Native Context Integration > Raw Prompt Capacity

Demo Time

"What can possibly go wrong?"

Moving out of static slides and straight into live environment operations.

- **Zero Net Safety:**
- **Deterministic Engine:**
- **Live Code**

Beyond Programming: Full SDLC Value

While engineers drive the IDE interface, the underlying metadata and agentic understanding pipeline provides continuous value to the complete software lifecycle.

QA & Testing

Automated alignment of test assertions directly to legacy source dependencies. Generation of target regression matrix scenarios dynamically.

DevOps & Infrastructure

Immediate operational parsing of environment anomalies, configuration mismatch logs, and active build path resolution schemas.

Product & Analytics

Translating ancient logic structures into clean feature requirements, mapping functional parameters without deep system exploration.

The AI From easy wins to holy grail

↑ Moving Upwards from Foundational Baselines to High-Velocity Application Delivery

Use cases - not really limited

Click Bob to Return to Main Page

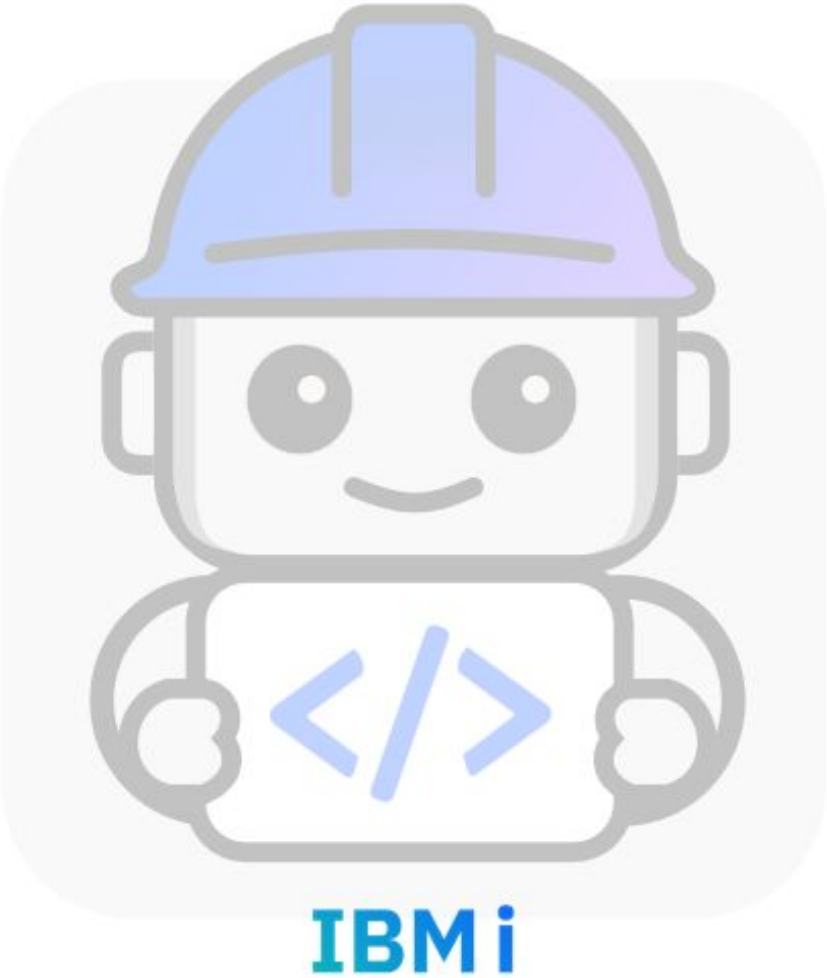
Bob and IBM i

Explain

1. Explain a Simple RPG Application
2. Explain a 1980 era ERP Application complete with Diagrams
3. Impact Analysis Report for File Level Access and Change

Refactor

1. Refactor Variable Names to be Self Describing
2. Extract Procedures and Create Service Program
3. Remove Un-Used Variables
4. Convert O-Spec to Print File



Transform

1. RPGII to Modern Free Format ILE RPG with Readable Variable Names
2. RPG Fixed to Free Conversion
3. Database - Replace Record Level Access Database Access with Embedded SQL Access
4. Transform 2Digit year to Real Date Field

Generate

1. Generate Comments in Free Form RPG Program
2. Generate a brand new Vechile Management App – SQL, RPG, Python
3. Literate Mode for Code Completion in Line
4. Create Unit Testcases Leveraging RPGUNIT

Use case – code explanation

- Turning silent IBM i code into business understanding
- No refactor. No rewrite. Just clarity

Use Case: Let AI Do the Boring Plumbing

Target: Generate Dynamic SQL in COBOL

Handling variable input fields matching @SNDSGNSPL.PF without building vulnerable or broken dynamic code strings manually.

- Build dynamic WHERE clause from parameters actually passed (ignore blank/zero).
- Open, execute, and loop through SQL query result rows.
- Pass processed records down to an "actual work" routine.
- Protect natively against SQL injection vulnerabilities.
- Centralize generic error handling & apply full ILE-style comments.

Why This is a Perfect AI Task

This is not creative business logic. **This is plumbing.** It is inherently ugly, highly repetitive, easy to get almost right, and mind-numbingly boring to review.

Machine's Responsibility

Let AI generate the boring, defensive code structures. If the programmatic boilerplate work has no creative soul, let the automated machine suffer through writing it.

Engineer's Responsibility

Let senior developers review correctness, safety, and business meaning. Shift developer fatigue away from boilerplate and onto high-level architecture decisions.

`Fits directly into IBM i-oriented tools, code refactoring loops, and asset workflows.`