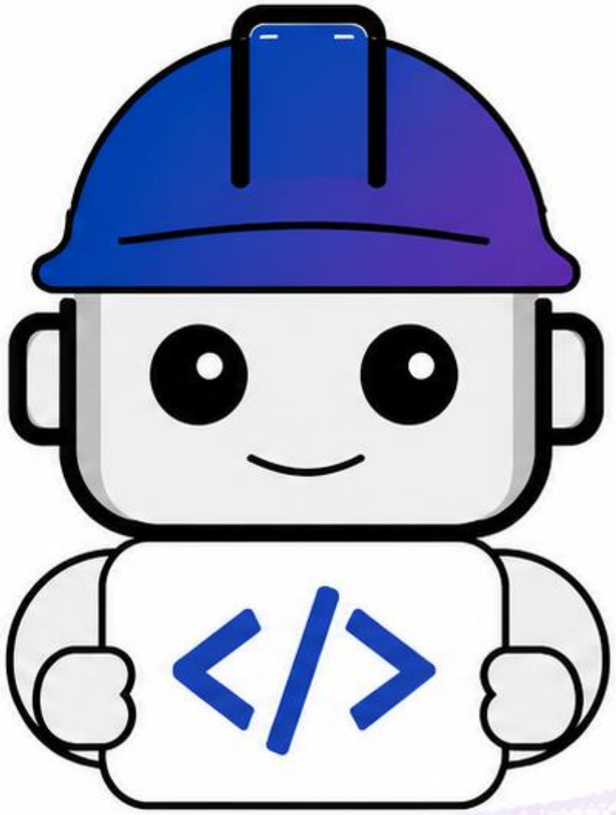
א.מ.תברמור

IBM Platinum Partner

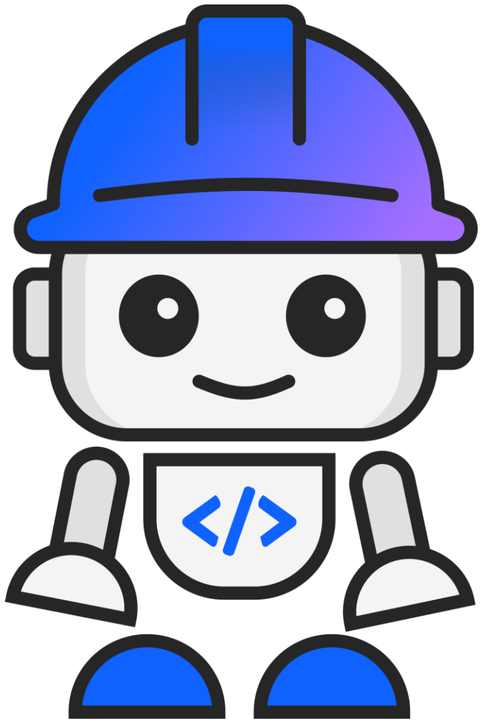
AI on IBMi & Mainframe

Tal Neeman – Data & AI Platform Engineer



Vibe coding **responsibly.**

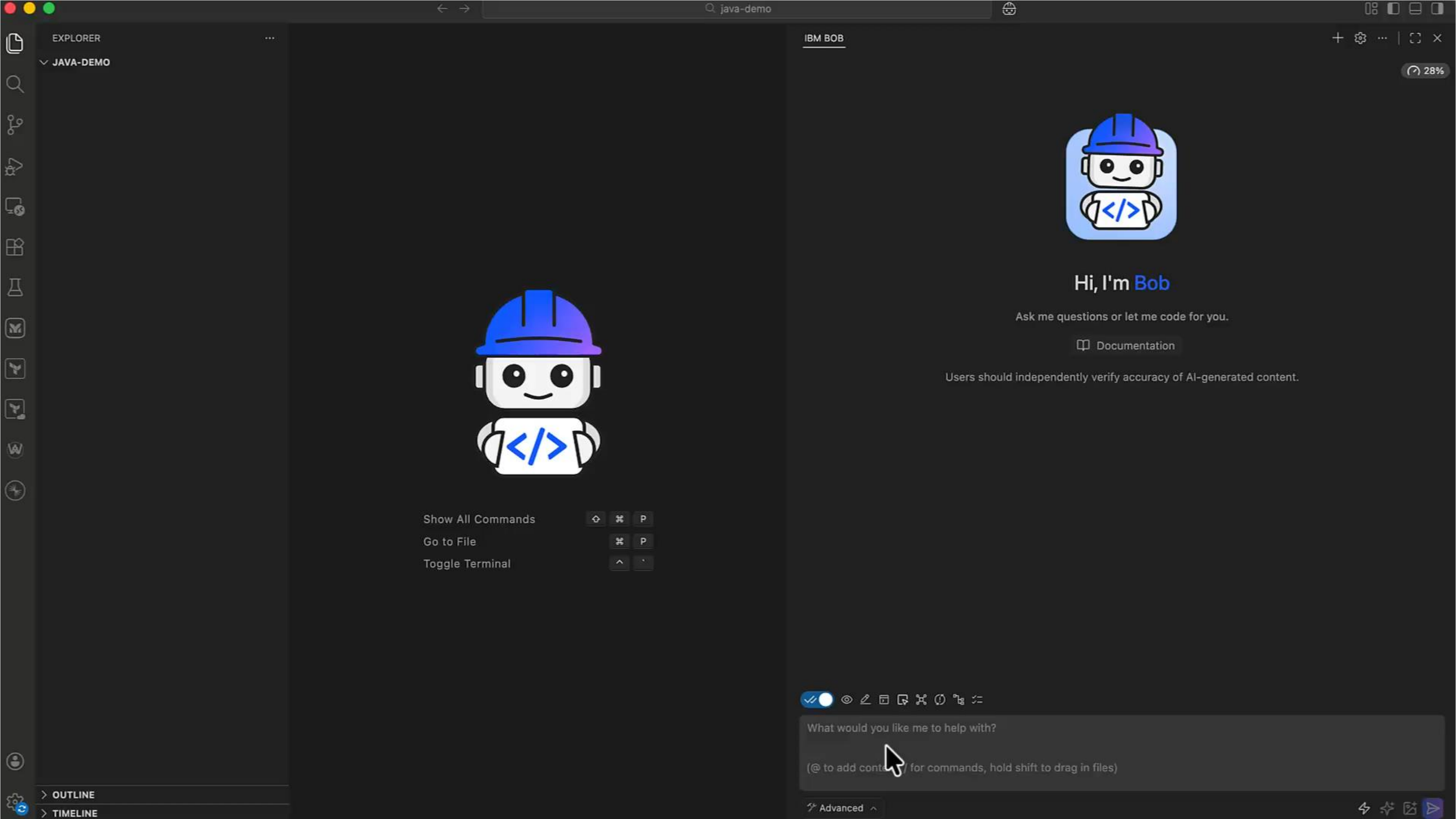
From AI hype to enterprise delivery, and what IBM Bob is built to do about it.



Tal Neeman
Data & AI Platform Architect, IBM

What is Vibe coding?

“Vibe coding” is a new and loosely-defined term in software development that refers to the practice of prompting AI tools to generate code rather than writing code manually.


java-demo
EXPLORER
JAVA-DEMO
Show All Commands
Go to File
Toggle Terminal
OUTLINE
TIMELINE
IBM BOB
28%
Hi, I'm Bob
Ask me questions or let me code for you.
Documentation
Users should independently verify accuracy of AI-generated content.
What would you like me to help with?
(@ to add cont / for commands, hold shift to drag in files)
Advanced

Vibe coding is ~~not~~ what most people think.

NOT — 01

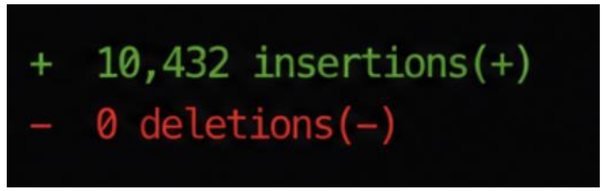
Prompting faster.

Output speed is not the skill. You can generate code faster than you can read it — that's the danger, not the goal.

NOT — 02

Shipping more code.

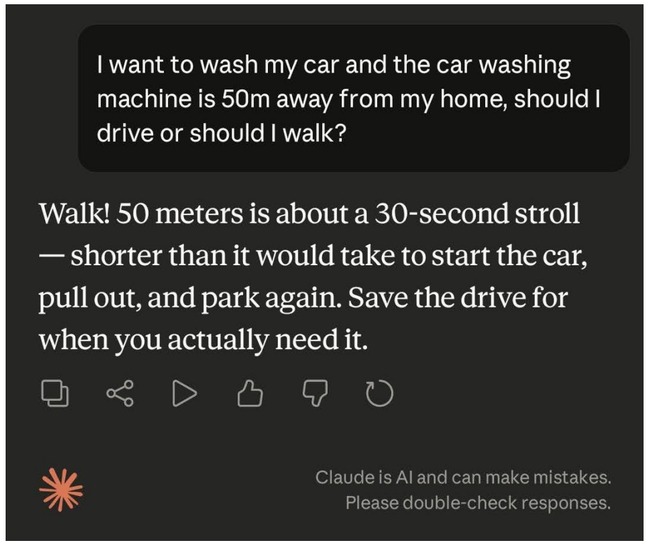
Code volume is a vanity metric. Production needs working software, not large diffs.



NOT — 03

Trusting the output.

If you can't verify it, you can't ship it. "AI generates → you trust → it deploys" is how AI bugs reach production.



AI gets it wrong, often.

FAILURE MODE 01

Speed without comprehension.

Output arrives faster than any human can read it. Review becomes performative.

FAILURE MODE 02

Hallucinated dependencies.

APIs that don't exist. Versions that don't match. Imports that look right and aren't.

FAILURE MODE 03

Vulnerabilities by default.

Insecure patterns reproduced from training data at the speed of generation.

FAILURE MODE 04

Tech debt, accelerated.

More code, more duplication, more interdependencies and no system that sees them.

So, is this just for side projects, or can enterprise actually use it?

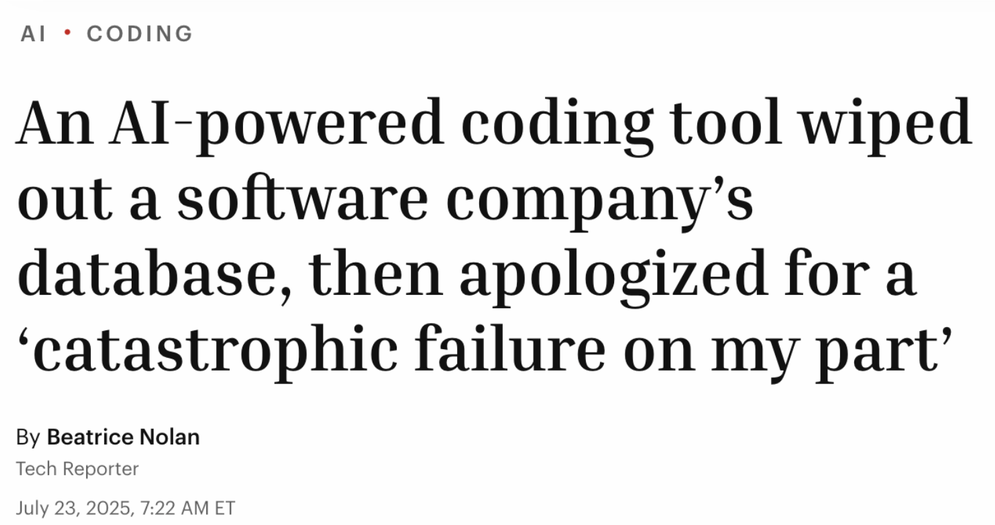
The security tab **always** comes due.

RISK 01 Secrets leak into prompts and repos.

RISK 02 OWASP-class vulnerabilities, shipped.

RISK 03 AI-native attacks — prompt injection, jailbreaks, model CVEs.

RISK 04 No audit trail for AI decisions.



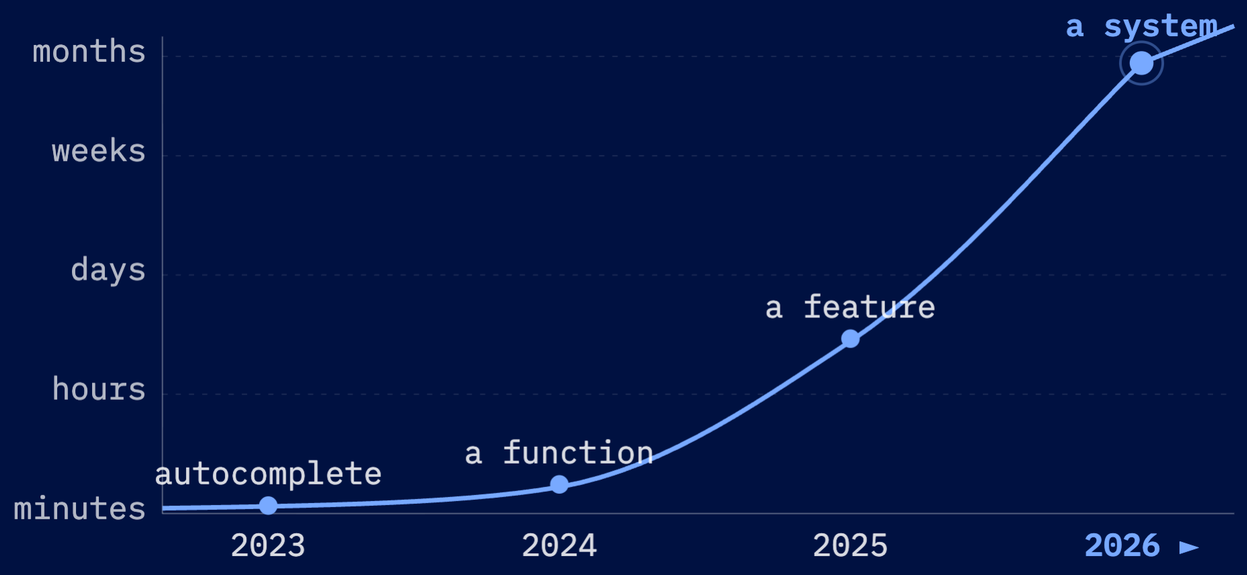
AI • CODING

An AI-powered coding tool wiped out a software company's database, then apologized for a 'catastrophic failure on my part'

By **Beatrice Nolan**
Tech Reporter
July 23, 2025, 7:22 AM ET

And yet, you can't **opt out**.

Model capability is climbing exponentially. Days a year ago are hours today. Weeks today will be days soon.



IBM, ON IBM

45%

productivity gain across IBM teams

$3M+

cost savings · Client Zero

80K

IBM developers · organic adoption

Forget the code. **Focus the product.**

A shift in abstraction level , not a shortcut. You're the director. AI is the most skilled contractor you've ever had — still in need of supervision.

`term coined by andrej karpathy · 2025`

We used to read the **assembly**.

Now we trust the abstraction — because we built verification above it. AI is at the same moment.

THEN · 1970s

asm

```asm
section .data
    message db "Hello, World!", 10
    length equ $ - message

section .text
    global _start

_start:
    ; write(stdout, message, length)
    mov rax, 1          ; syscall: write
    mov rdi, 1          ; file descriptor: stdout
    mov rsi, message    ; pointer to message
    mov rdx, length     ; message length
    syscall

    ; exit(0)
    mov rax, 60         ; syscall: exit
    xor rdi, rdi        ; status = 0
    syscall
```

Distrust of the new abstraction. Programmers read the output to verify it.

NOW · today

C++

```cpp
#include <iostream>

int main() {
    std::cout << "Hello, World!" << std::endl;
    return 0;
}
```

Nobody reads assembly. We trust tests, types, linters, CI above it.

Every leader manages what they **can't directly inspect**.

01 · EDITOR-IN-CHIEF

Doesn't write every article.

VERIFIES THROUGH

editorial standards · fact-checking · drafts & review

02 · SYMPHONY CONDUCTOR

Doesn't play every instrument.

VERIFIES THROUGH

the score · rehearsals · the sound in the room

03 · HEAD CHEF

Doesn't cook every plate.

VERIFIES THROUGH

recipes · plating standards · tasting the line

Each one finds a verifiable abstraction. Vibe coding is the same problem at a new scale.

Tech debt has **no good verifier**.

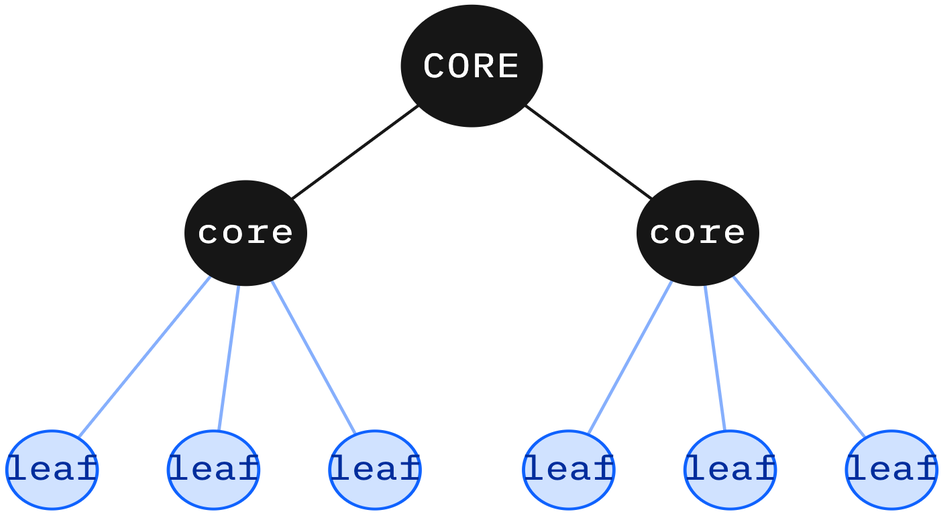
Unlike functional bugs, tech debt has no test suite, no green light, no automated check.

Only technical people can spot it. And reviewing AI-generated code for quality, duplication and architectural drift is slow.

functional bugs	TESTS CATCH IT
security flaws	SCANNERS CATCH IT
style & lint	LINTERS CATCH IT
tech debt	**HUMANS ONLY · SLOW**

The enterprise problem is not generation. **It's verification.**

Two kinds of components.
Two ways to build them.



CORES · cautiously · with humans
LEAVES · autonomously · by agents

● CORES — SUPERVISED

Architecture, data, integrations. Still vibe-coded — but cautiously, with humans in the loop on every step.

◯ LEAVES — AUTONOMOUS

Isolated features. Bounded blast radius. Built by background agents — triggered by a git issue or running inside the CI/CD pipeline.

How to vibe code **well.**

FOR THE CORES

01 Context before code.

15 minutes of planning prevents 3 hours of debugging.

02 Ask the right questions.

Domain knowledge still matters — challenge the AI.

FOR THE LEAVES

03 Verification tests.

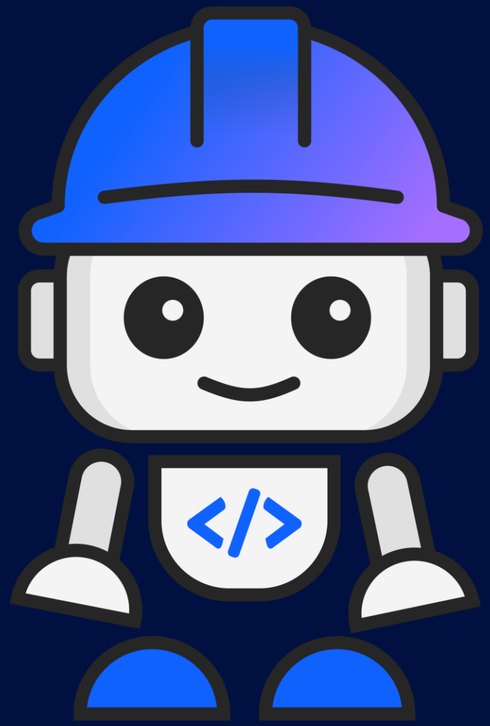
An automated layer that tests the system without a human in the loop.

04 Human-readable outputs.

Dashboards, audit trails, reports — verifiable without reading code.

An agentic SDLC partner

Meet Bob.



A development partner that plans, executes, validates, and governs changes across your SDLC — auditable at every step.

Principle 01 · context before code

Tell Bob **what you're doing** — before he writes anything.

PLANNING MODE

Design first. Code second.

Bob produces a plan you can review before a single line changes.

CUSTOM MODES

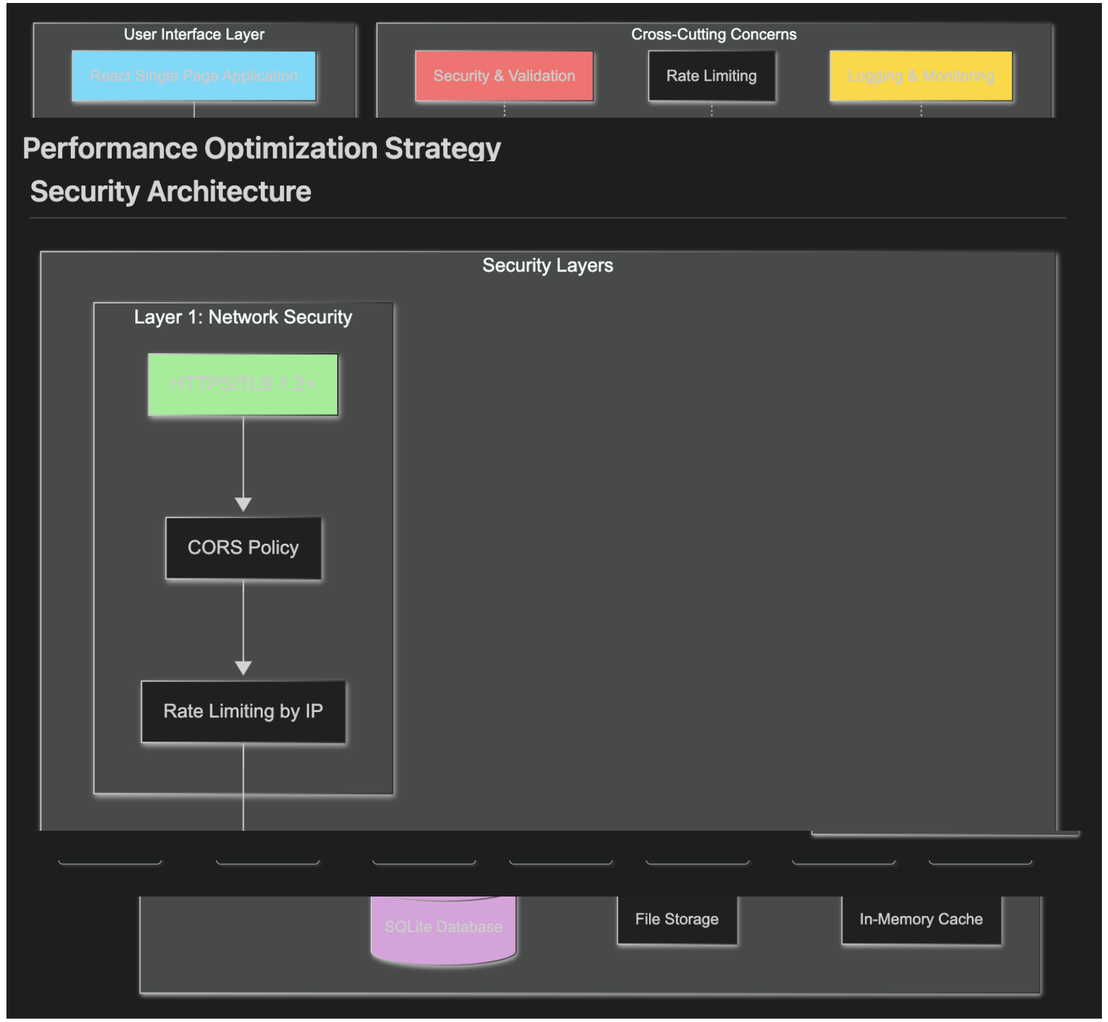
Ask · Plan · Code · Advanced · your own.

Each mode constrains what Bob does and how he does it.

RULES & .BOBIGNORE

Your standards, shipped with every dev.

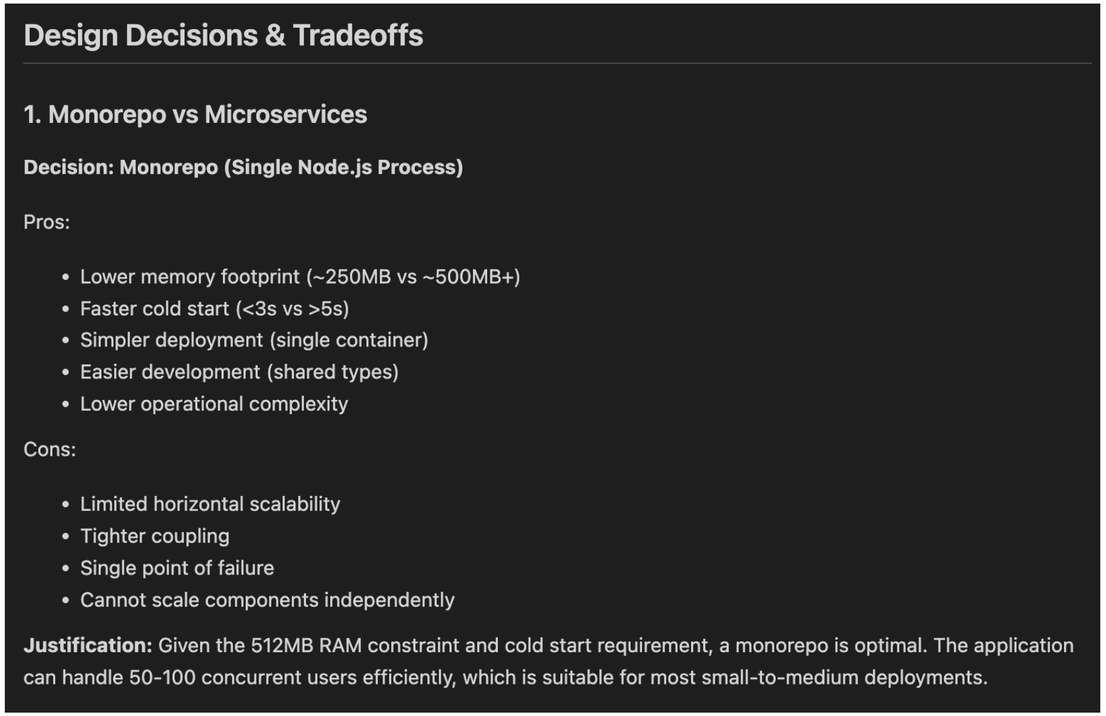
Project-wide and mode-specific rules. File boundaries Bob won't cross.



Principle 02 · ask the right questions

Bob **asks you back.**

And produces extensive plans — with the architectural *why* behind every step.



Design Decisions & Tradeoffs

1. Monorepo vs Microservices

Decision: Monorepo (Single Node.js Process)

Pros:

- Lower memory footprint (~250MB vs ~500MB+)
- Faster cold start (<3s vs >5s)
- Simpler deployment (single container)
- Easier development (shared types)
- Lower operational complexity

Cons:

- Limited horizontal scalability
- Tighter coupling
- Single point of failure
- Cannot scale components independently

Justification: Given the 512MB RAM constraint and cold start requirement, a monorepo is optimal. The application can handle 50-100 concurrent users efficiently, which is suitable for most small-to-medium deployments.

Clarifies before acting.

Surfaces the questions you'd have asked — if you'd thought of them.

Architecture rationale, step by step.

Each decision is justified — so the plan is reviewable, not just runnable.

Confidence before execution.

You approve a plan you understand — not a black box.

Principles 03 & 04 · verification + outputs

Tests Bob writes. Findings Bob explains.

VERIFICATION TESTS

Auto-generated. Run automatically.

Bob writes tests as he writes code — an automated layer that verifies the system without a human in the loop.

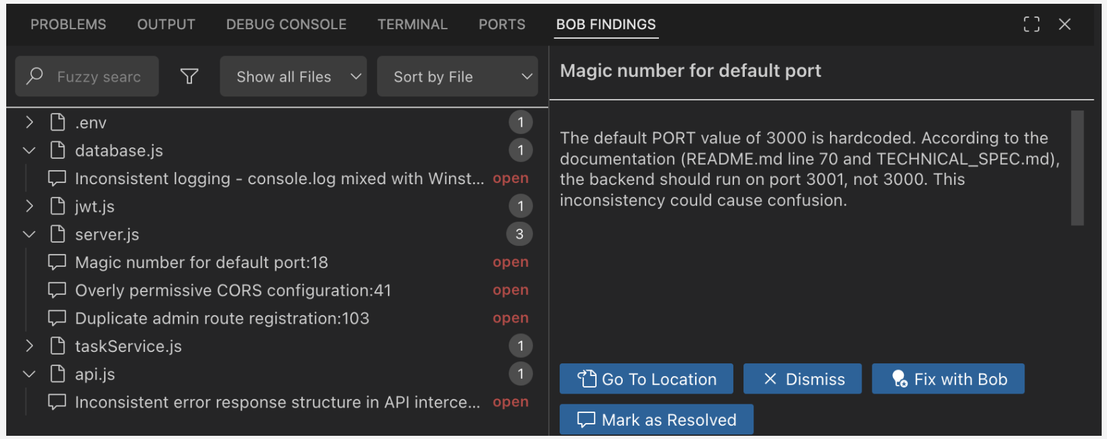
95%
test coverage

−70%
QA effort

BOB FINDINGS

Surfaces problems. Explains the fix.

Bob audits the code, flags issues — duplication, drift, smells, bugs — and walks you through how to fix them in plain language.



Security, **built in.**

Security embedded in the authoring loop — not bolted on at the end, not gated only at PR, but enforced continuously across IDE → CI → deploy.

STAGE 01

Plan

Architecture intent traced; risks surfaced before any code is written.

STAGE 02

Develop

Secret scanning, .bobignore boundaries, sensitive-data redaction.

STAGE 03

Integrate

OWASP / CWE classification on every PR; required reviewers and blocking conditions.

STAGE 04

Validate

CI/CD policy gates — security scans, coverage, license checks before merge.

STAGE 05

Deploy & operate

AI-aware runtime — prompt-injection blocking, model-CVE detection, audit telemetry.

COMPLIANCE — OUT OF THE BOX

SOC 2 | FedRAMP | HIPAA | PCI

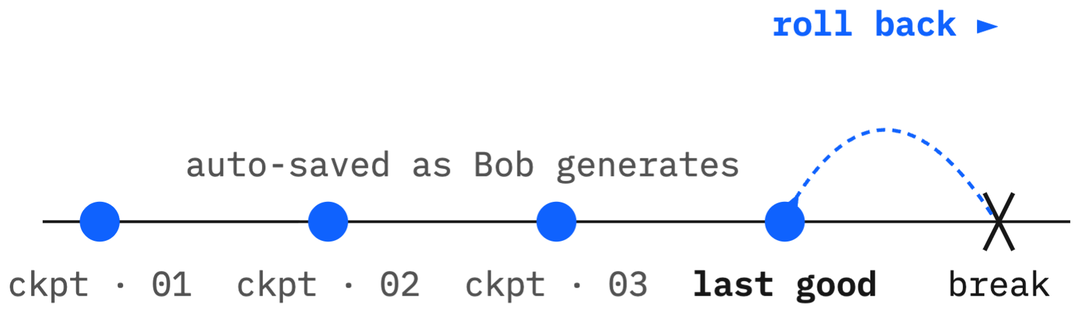
Vulnerability Details:

- **OWASP Category**: A03:2021 - Injection
- **Severity**: Critical
- **Root Cause**: Direct shell command execution with user input
- **Weak Validation**: Regex uses `.find()` instead of `.matches()`, allowing trailing content

When something goes wrong — and it will

Checkpoints.

Sometimes restarting from a working state beats debugging a hidden one.



Auto-saved as Bob generates.

Every meaningful step becomes a restore point.

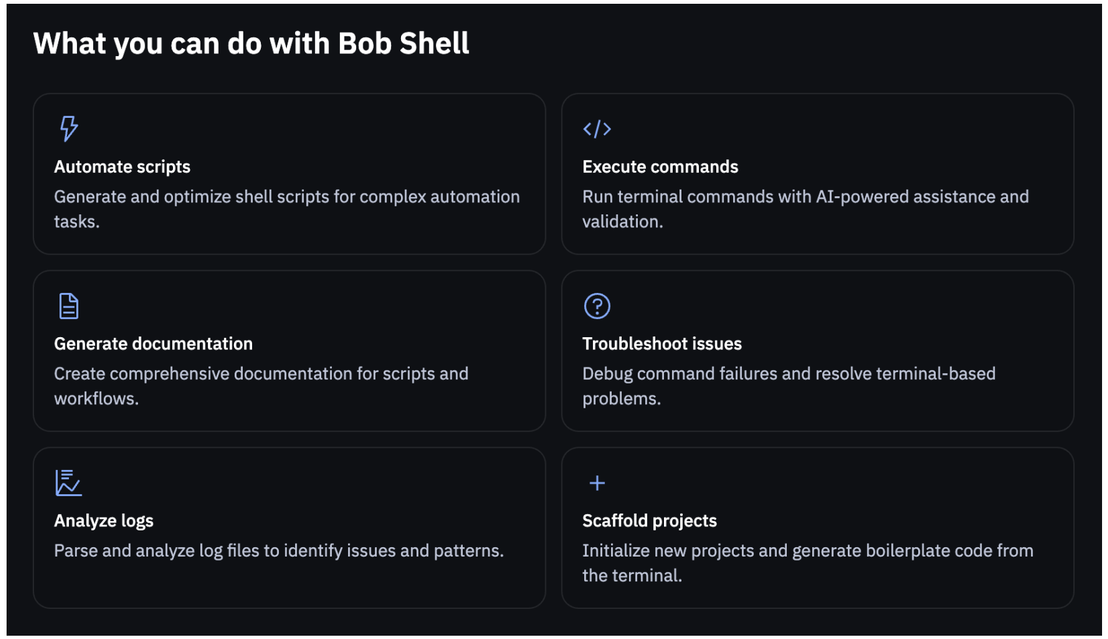
Roll back any time.

If a large diff went wrong, return to the last working state — instead of hunting the bug.

For the leaves

Bob Shell — Bob outside the IDE.

Run Bob from the terminal, inside CI/CD, or as a background agent listening to git issues.



Terminal-native.

Same context awareness as the IDE.

CI/CD-embedded.

Policy gates. Auditable runs. No surprises in the pipeline.

Background agent.

Triggered by a git issue — works while you sleep.

Fluent in the languages enterprises **actually** run.

LANGUAGES & PLATFORMS

COBOL	RPG
IBM i / AS400	Z / mainframe
C, C++	Python
Java	Node

The old languages most AI tools have never seen — Bob knows them, and the mainframes / IBM i they run on.

DEPLOYMENT

SaaS

available today

IBM Cloud-hosted model gateway.

On-prem / Sovereign

roadmap

Strict residency · enterprise KMS / HSM.

Air-gapped

roadmap

Isolated networks · regulated industries.

Z / Power native

roadmap

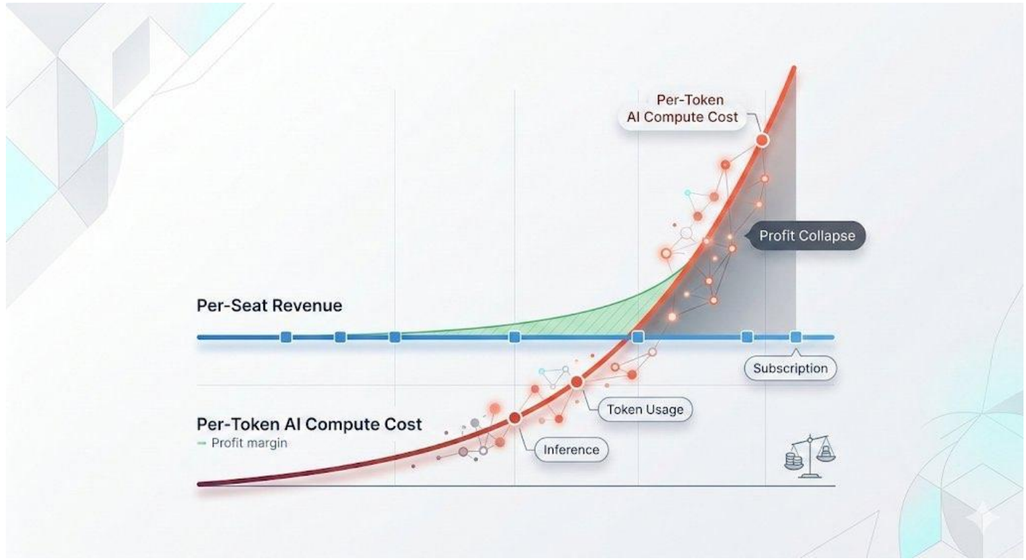
Mainframe & IBM i environments.

Costs, in the open

Bobalytics — clear costs, real productivity.

Token spend per user, team, and workflow — alongside the productivity it bought.

45%	$3M+	~40%	10×
PRODUCTIVITY GAIN	COST SAVINGS	AI COST SAVED	DOCS FASTER
across IBM teams	Client Zero adoption	routing · caching · edge RAG	100% codebase coverage



Cost optimization

Cost for AI tools is real-time engineering signal that must be actively managed.

But model behavior and resource consumption are often opaque and difficult to control.

IBM Bob provides visibility, control, and optimization of cost across workflows, enabling teams to scale AI usage responsibly.

With IBM Bob, you have a decisioning system that optimizes cost at every step

Task classification:
What is the nature of the task?

Complex + multi-file
↓
Simple explanation
↓
Bulk + repetitive task
↓
Sensitive + regulated

Intelligent model routing:
How should this task be executed?

High- reasoning
↓
Cost-efficient
↓
Other cost criteria

Admin cost controls
Is this within expected cost & value?

Within budget
↓
Approaching budget
↓
Exceeds threshold
↓
Pause or approve

Attribution & outcome mapping
Was the cost justified?

High value: scale
↓
Low value: optimize
↓
Repeatable

Workflow optimization
What should change going forward?

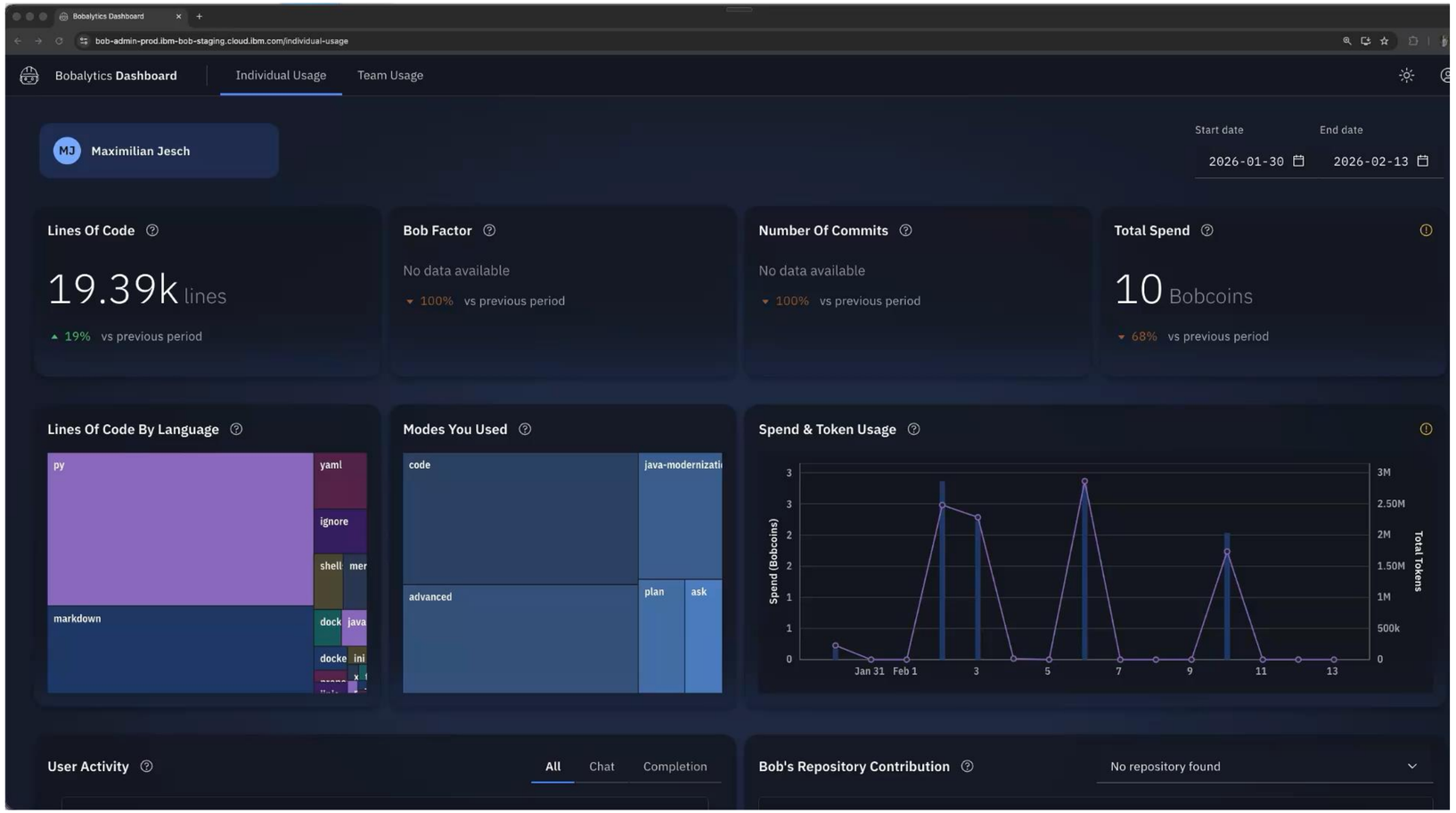
Automate workflow
↓
Reroute future tasks
↓
Refactor process
↓
Eliminate waste

Cost optimization

Cost for AI tools is real-time engineering signal that must be actively managed.

But model behavior and resource consumption are often opaque and difficult to control.

IBM Bob provides visibility, control, and optimization of cost across workflows, enabling teams to scale AI usage responsibly.



The takeaway

So, Forget the code?
Never forget the product.

TRY BOB

bob.ibm.com

TALK TO US

Q&A · workshop · pilot