Moderne

Moderne Prethink for the mainframe

Accelerate mainframe modernization with deterministic agent context.

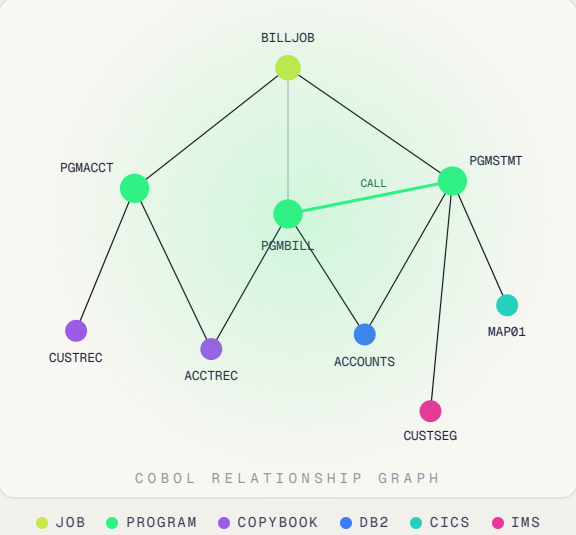
You want the technical debt gone and the applications cloud-native. But a single mainframe estate runs to tens of millions of lines, and the largest context window holds maybe a hundred thousand of them. **An agent sees a fraction of a percent of the system at once.** What it cannot see, it infers.

Moderne Prethink is the vendor-neutral impact-analysis layer for AI-driven mainframe modernization. It builds a compiler-accurate map of your mainframe estate: architecture, business rules, data access, and lineage. Every agent and every engineer starts from the same deterministic data, with the shape of the whole system in view.

Moderne Prethink vs IBM ADDI. ADDI delivers the analysis to a viewer for engineers to browse, refreshed on import. Prethink delivers it to the repo as open files, regenerated from the code and ready for an agent on the first prompt.

Every dependency, typed and resolved

- **Shared copybooks surface immediately.**
- **Contended resources are visible before you plan.**
- **Runtime paths are typed, not guessed.**
- **Missing dependencies are reported as findings.**



One recipe bundle, source to committed context

01.

Parse

COBOL, JCL, and the rest of the estate's members are parsed into compiler-accurate LST code models, with copybooks, Control-M schedules, field declarations, `CALL` targets, CICS commands, DL/I calls, and embedded SQL all resolved.

02.

Analyze

The Prethink mainframe recipe bundle runs across the LSTs, collecting the data tables and analyzing the estate. Every row is read from resolved, fully type-attributed code: nothing sampled, nothing inferred.

`UpdateMainframePrethinkContext`

03.

Commit

All data files and analysis are committed and versioned under `.moderne/context/` for agents to access when they open a session. Includes `mainframe-architecture.json` (FINOS CALM), `findings-report.md`, `column-lineage.json`, `*.csv` (raw data).

Moderne runs against git, not on z/OS. *Source reaches the repository through bidirectional sync, and the analysis runs entirely off-mainframe.*

Partner with Moderne · partnerships@moderne.io · www.moderne.ai

Moderne

What Prethink analyzes in mainframe code

Architecture

JCL jobs, programs, data sets, DB2 tables, CICS resources, and IMS segments, and how they reach each other.

Business rules

What each program decides, the conditions that decide it, and where those conditions contradict each other.

Data access

Which programs create, read, update and delete which files, queues, DB2 tables and IMS segments, and the JCL steps that bind them to data sets.

Data lineage

Where data comes from and where it ends up, at column granularity: which field feeds which column on which data set, and the route between them.

Copybooks

The portfolio's shared snippets: what each copybook declares, who uses which of its fields, and where the copies disagree.

Program metrics

What each program costs to change: size, complexity, maintainability, and dead data. Triage for deciding which programs to move first.

Migration blockers

CICS and IMS constructs that need a design decision before a program can move, and the API surface behind them.

Ranked findings

Everything found wrong, ranked, with evidence. Critical means code that cannot do what it was written to do.

What agents can discover from Prethink

With Prethink context in the repository, an agent gets answers from facts that let it move forward with its modernization work.

Which programs are hardest to move?

Size, coupling and maintainability per program, flagged at thresholds. Four CICS API groups is a very different proposition from twelve.

What does this program decide?

The rules it applies, as a condition and action directly from the source, including rules that are contradictory or can't be reached.

Can these two programs be separated?

Not until someone decides who owns the resource they both write.

Where does this field's data go?

Column-level lineage from field to data set, with the route through the program.

What breaks if this copybook changes?

The short list of programs that use the fields, not the long list that copy the book.

What needs a design decision rather than a translation?

Control flow that leaves and doesn't come back. State that outlives a transaction. Work bound to one z/OS region. Run-time resource names. Checkpoint and restart.