Moderne Prethink for the mainframe

Accelerate mainframe modernization with deterministic agent context.

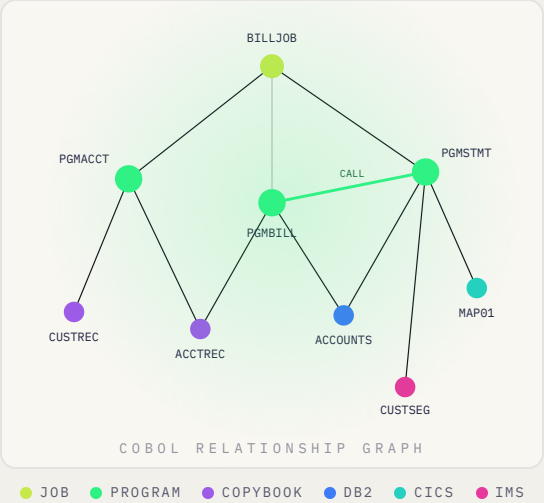
You want technical debt gone and applications cloud-native. But a single mainframe estate can run to tens of millions of lines, while even the largest context windows only hold about a million tokens. **That means an agent sees a fraction of a percent of the system at any one time.** What it cannot see, 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, resolved across repositories and down to the individual field. Every agent and every engineer starts from the same deterministic data, with the shape of the whole system in view.

THE MAP

Every dependency, typed and resolved

- **Shared copybooks surface immediately.** When two programs copy the same book, the cost of changing it is the short list that uses the affected fields, not the long list that copies it.
- **Contended resources are visible before you plan.** Two programs writing one DB2 table cannot be separated until someone decides who owns it.
- **Runtime paths are typed, not guessed.** A CICS map, an IMS segment and a DB2 table are distinct node types with distinct edges, resolved from the code that reaches them.
- **Missing dependencies are reported as findings.** A name the source uses that no analyzed repository defines is an answer, not a blank.



HOW IT RUNS

One recipe bundle, source to committed context

01

Parse

COBOL and JCL sources are parsed into their compiler-accurate LST code models. Copybooks resolve to their library members, fields to their declarations, `CALL` targets across program boundaries, and CICS commands, DL/I calls and embedded SQL through the COBOL that issues them.

→

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 provided under `.moderne/context/`, reviewed, committed, and versioned. Agents can access the files when they open a session.

`.moderne/context/`

`mainframe-architecture.json`	A FINOS CALM document. What runs what, what holds data, and every runtime path between jobs, programs, data sets, DB2 tables, CICS resources and IMS segments.
`findings-report.md`	Everything the analyses found wrong, ranked 1 to 4, each row carrying the evidence it came from and the file it lives in.
`column-lineage.json`	Where data comes from and where it ends up, traced at the level of individual COBOL fields. It follows a field through the program to the file, queue, DB2 table, IMS segment or terminal it lands on. Written as an OpenLineage document.
`*.csv`	All the raw data collected from the codebase, one set per topic, ready to query.

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

CONTEXT

What Prethink analyzes in mainframe code

Architecture

The portfolio's architecture, including 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 job steps that bind them to real data sets. Who touches what, and how.

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, and what depends on them, including what each copybook declares, who uses which of its fields, and where the copies disagree.

Program metrics

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

Migration blockers

What stands between a program and a platform that isn't z/OS. CICS and IMS constructs that need a design decision before a program can move, and the API surface behind them.

Ranked findings

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

IN PRACTICE

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?
Decision tables per program, with the rules that contradict each other or that nothing can reach flagged.

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.

Who else reads this data set?
One query against the edge list.

What can we leave behind?
The fields that nothing reads.

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.

If you are familiar with IBM ADDI, this is similar coverage delivered differently. ADDI's analysis lands in a project that engineers explore through a viewer, refreshed when someone imports it. Prethink regenerates from the code and provides open-format files for an agent to read immediately on starting a task.