1 Using AI Safely in Mission-Critical COBOL Modernization

Generative AI can provide valuable assistance during a COBOL modernization project. It can help developers understand unfamiliar programs, generate documentation, identify dependencies, suggest tests, and explain complex sections of legacy code.

However, explaining or rewriting individual programs is different from translating an entire production application. Enterprise modernization requires consistent handling of business logic, control flow, data definitions, files, databases, transaction-processing APIs, error paths, and interactions between programs.

This article examines where generative AI can add value, where caution is required, and how AI-assisted analysis can be combined with deterministic translation.

1.1 Where Generative AI Can Add Value

Generative AI is most useful where modernization requires interpretation, summarisation, or developer assistance. It can help with:

- explaining unfamiliar COBOL and business logic;
- generating program and paragraph documentation;
- supporting application discovery and dependency analysis;
- suggesting test scenarios and boundary conditions; and
- assisting developers with translated or newly developed code.

1.2 Where Generative AI Requires Caution

Generative AI is valuable for code explanation and developer assistance, but enterprise application modernization presents additional challenges. The generated output may vary between model versions or prompts, and an individual COBOL program often cannot be translated correctly without the surrounding application context, including copybooks, called programs, databases, files, screen definitions, JCL, and transaction-processing services.

Enterprise COBOL applications also rely on platform-specific technologies, such as CICS, IMS, DB2 embedded SQL, VSAM, JCL, Unisys DMS-II, and HPE NonStop services, whose behaviour must be reproduced consistently across the entire application.

Readable or compilable Java or C# code does not, by itself, demonstrate functional equivalence. Validation should confirm that file processing, database updates, numeric precision, transaction behaviour, return codes, exception handling, and program interactions remain consistent with the original COBOL application. AI-generated code therefore typically requires expert review and execution-based testing before production deployment.

1.3 What Deterministic Translation Provides

Deterministic translation applies defined rules consistently across the complete application. Its principal advantages are:

- repeatable output from the same COBOL source;
- consistent mappings for data, files, databases, screens, and transactions;
- controlled regeneration when COBOL programs change; and
- a common architecture across large application portfolios.

Learn more about how deterministic translation compares with generative AI in Claude Code vs Deterministic COBOL Modernization .

1.4 Using AI and Deterministic Translation Together

Generative AI and deterministic translation address different parts of modernization. Deterministic rules can be used for executable application conversion, while AI supports documentation, code explanation, discovery, test preparation, and developer productivity.

This allows AI to be used where interpretation is valuable, while deterministic processing provides repeatability, consistency, and controlled regeneration across the complete application.

1.5 Choosing the Appropriate Approach

The best modernization approach depends on the size, complexity, and objectives of the project. AI-assisted development may suit smaller or isolated tasks, while large mission-critical applications generally require a controlled, repeatable process.

When evaluating an approach, consider:

- whether the complete application can be processed;
- whether results are repeatable;
- how platform-specific technologies are handled;
- how functional equivalence is demonstrated; and
- whether the application can be regenerated after future COBOL changes.

A representative pilot should use real application programs, execute business scenarios, and compare the results with the original COBOL application.

1.6 Practical Evaluation of Claude Code

SoftwareMining has evaluated Claude Code using COBOL applications of varying size and complexity, assessing generated code, application context, repeatability, and suitability for enterprise modernization.

See Claude Code vs Deterministic COBOL Modernization .

1.7 Testing Translated Applications

Successful modernization should be demonstrated through execution-based testing using representative business scenarios and comparison with the original COBOL application.
See Testing and Functional Equivalence for Migrated COBOL Applications .

1.8 Conclusion

Generative AI and deterministic translation are complementary rather than competing technologies. AI is valuable for analysis, documentation, discovery, and developer assistance. Deterministic translation provides the repeatability, consistency, and auditability needed for enterprise application conversion. Used together, they allow organisations to benefit from AI-assisted productivity while retaining the controlled process required for mission-critical modernization.

1.9 Questions to Ask Before Using AI for COBOL Modernization

- Can the complete application be processed, including copybooks, JCL, screens, files, and database access?
- Are results repeatable from the same source?
- How is functional equivalence demonstrated?
- Can the application be regenerated after COBOL changes?
- Which parts are translated deterministically and which parts rely on AI assistance?