Claude Code modernizes a legacy COBOL codebase

4 MIN · YOUTUBE · HTTPS://WWW.YOUTUBE.COM/WATCH?V=OWMU0PYYZBC
https://www.youtube.com/watch?v=OwMu0pyYZBc

SUMMARY

Developers can leverage Cloud Code to modernize a legacy Cobalt codebase, such as a credit card management system, by first documenting the existing code and then migrating it to a modern language like Java. The process involves creating a detailed analysis of the codebase, generating comprehensive documentation, and executing a structured migration plan that ensures fidelity in functionality.

- *Phase 1 focuses on discovery and documentation, addressing the lack of documentation in legacy codebases.*
- *Cloud Code utilizes a specialized sub-agent to analyze the architecture and create a to-do list for processing files.*
- *Documentation includes detailed business workflows, dependency mapping, and visual diagrams of processes.*
- *Phase 2 involves migrating core features to Java, with a structured five-phase migration plan.*
- *The migration plan includes creating project structures, translating data models, building IO layers, converting business logic, and establishing dual test harnesses.*
- *Verification ensures the new Java code maintains bit-for-bit fidelity with the original Cobalt code.*
- *The techniques demonstrated are scalable for larger legacy systems, enhancing developer efficiency and confidence in modernization efforts.*

Let's explore how developers can use Cloud Code to modernize a Cobalt codebase. For the purposes of this demo, we'll use AWS's mainframe modernization demo repository. This is a medium-siz credit card management system with around 100 files, including Cobalt programs, copy books, and JCL scripts. Phase one, discovery and documentation. Our sample Cobalt codebase has almost no documentation. This is of course common with legacy code bases where critical business logic and regulatory requirements are embedded within undocumented code. The developers who wrote the code have long since left the organization and developers familiar with Cobalt are hard to hire. We first create a specialized sub aent using cloud code/ aent command. This was our Cobalt documentation expert in translator. Sub aents can be invoked by cloud code in parallel and they operate with their own isolated context windows to avoid polluting the main thread. We enabled thinking mode and asked cloud code to analyze the architecture of the codebase. Cloud code created a to-do list of all 94 files and tracked its progress to ensure no files were processed twice and nothing was missed.

The documentation Claude produced went beyond simple code comments. For example, let's look at the interest calculation program CBAC4C. It extracted the complete business workflow, how the program reads transaction category balances, looks up interest rates by account group, applies business rules for fallback rates, and updates account records. Claude did this for each file, but also created two memory files as plain text. Catalog.ext text translates cryptic names like CBAC04C into interest calculator batch program relationships.ext maps every dependency using a simple pipe delimited format. Using these indices, Claude then generated

mermaid diagrams, a complete map of the daily batch processing workflow showing how the data flows from transaction input through posting interest calculation and finally to customer statements.

In this demo, Cloud Code ran continuously for an hour to draft over 100 pages of documentation. But Claude Code is capable of running for over 30 hours autonomously, and the techniques used here scale to much, much larger code bases. Phase 2, migration and verification. After thoroughly documenting the Cobalt codebase, we asked Claude to migrate one of its core features to Java. We switched to planning mode to ensure Claude would think through the entire migration strategy without prematurely editing files. Claude analyzed the program formerly known as CBAC4C and identified complex cobalt patterns like line break processing and multifile coordination. Claude developed a migration plan for this feature with five phases. One, create the project structure. Two, translate data models from copy books to Java classes. Three, build the IO layer compatible with the original file formats. Four, convert business logic while preserving Cobalt specific behaviors. And finally, create a dual test harness. One using GNU Cobalt 3.2.0 for the original codebase and one in Java 17. The resulting Java code went beyond a simple syntax translation. Claude created proper Java classes with appropriate design patterns, error handling, and logging.

idiomatic Java that a modern development team would actually maintain. Next was verification to ensure that the new Java code worked the same as the Cobalt code it was replacing. Claude created multiple test data files and ran them against both the original Cobalt and the new programs. The verification compared not just final outputs but intermediate calculations, file rights and data transformations. The result was perfect bit forbit fidelity. Every calculation, business rule, and edge case was preserved. Of course, this demo application is far smaller than your legacy Cobalt codebases, but all the techniques here are scalable. Claude Code will empower your developers to modernize code bases with confidence and efficiency that simply would have been impossible just 12 months ago.