

Playbook: COBOL in 2026

Modernization options without stopping operations

COBOL still runs critical billing, payroll, accounting and transaction processes at banks, insurers, credit unions and public agencies. The challenge is not the language itself, but knowledge concentrated in a few people, integration with digital channels, and the risk of touching something that cannot fail. This playbook helps you pick a path and move forward in stages.

1. Why it matters now

- COBOL is a living standard: the current edition is ISO/IEC 1989:2023.
- It is still widely used in mainframe applications, such as large-scale batch and transaction processing.
- Talent is a recognized risk: in a 2019 report on legacy systems, the U.S. Government Accountability Office (GAO) noted a dwindling number of people with the skills needed to support COBOL.
- In 2020, during the pandemic, several U.S. states reported a shortage of COBOL programmers for their unemployment benefit systems.

The question is not whether COBOL works, but how much risk you accept by depending on a few people and on processes that are hard to change.

2. Signs it is time to act

- Business rules live in the code and in the memory of a few people.
- Every regulatory or product change takes too long for fear of breaking the nightly batch.
- Digital channels (app, portal, APIs) need real-time data and currently rely on extracts or flat files.
- Documentation is out of date and the code is the only source of truth.
- Testing is manual and deployments are infrequent.
- Platform conditions (contracts, licenses, infrastructure) are forcing a decision.
- Auditors ask for change traceability and controls that the current process struggles to provide.

3. Options and when each one fits

No option is better in the abstract. Decide per business capability (interest, billing, payroll), not for the whole system: each part can take a different path.

Keep it and modernize how you work

When it fits: the platform is stable and the main problem is how the system gets changed. You add version control, automated tests, documented rules and current development tooling around the existing COBOL.

Trade-offs: it lowers operational risk and people dependency, but does not change the platform or the need for specialized talent.

Wrap with APIs

When it fits: digital channels need transactions or calculations that already work well. You expose services that call the existing programs.

Trade-offs: fast, reversible integration; the logic stays in COBOL, and the API layer must be designed for volume, security and batch windows.

Rehost (move with minimal change)

When it fits: the priority is leaving the current infrastructure while touching the code as little as possible, moving workloads to a compatible environment that runs COBOL.

Trade-offs: the code is still COBOL; you must validate performance, batch, security and everything the source platform handled (transactions, JCL, files).

Automated conversion (replatform)

When it fits: the code is consistent and you want to reach a modern language, such as Java or C#, without redesigning everything. Tools convert code and data structures.

Trade-offs: the result can end up as "COBOL written in another language", hard to maintain unless refactored later; it demands rigorous equivalence testing.

Incremental refactor (strangler fig pattern)

When it fits: the core is critical and will keep evolving. Capabilities are replaced one at a time behind a facade that routes each request to the old or the new system, with an anti-corruption layer that isolates the legacy model.

Trade-offs: lower risk per step and early value, but it requires transitional architecture, data synchronization, and the discipline to actually reach decommission.

Rewrite or replace with a product

When it fits: the process has fundamentally changed, or an off-the-shelf product covers the need with reasonable adjustments.

Trade-offs: the most uncertainty: undocumented calculations are easily lost, and a single cutover concentrates the risk.

Combining options is usually wiser than picking one

For example: first improve how you work (versioning and tests), wrap what digital channels need with APIs while rules get documented, and then replace capabilities one at a time with the strangler fig pattern. Each step leaves operations in a stable state.

4. A phased path

1 Assessment

Inventory of programs, copybooks, batch jobs and JCL, transactions, files, databases and interfaces. Identify critical business rules; AI can help read the code, always with human validation.

Deliverable: dependency map, risks, and a recommended path per business capability.

2 Pilot

A small, loosely coupled capability (for example, an inquiry exposed through an API or an isolated calculation), run in parallel with COBOL while comparing results.

Deliverable: pilot in production, agreed equivalence criteria, and a validated architecture.

3 Incremental migration

Capability by capability, ordered by value and risk, with synchronized data, characterization tests, and cutover windows agreed with operations.

Deliverable: each iteration adds migrated capabilities, documented rules and automated tests.

4 Decommission

Once nothing depends on the legacy program: controlled shutdown, archiving of historical data per regulatory requirements, and removal of licenses and infrastructure.

Deliverable: decommission sign-off and documentation for the new solution.

Illustrative example (fictitious)

Demo Credit Union calculates interest with a nightly COBOL program. It first exposes balance inquiries as an API for its mobile app. It then reimplements the calculation in .NET, runs it in parallel with COBOL over several closing cycles and, once results match under the agreed criteria, retires the COBOL program.

5. Risks and how to mitigate them

Risk	How to mitigate it
Hidden business rules (calculations, rounding, special cases)	Characterization tests with representative, anonymized data; validation with business owners; document every migrated rule.
Decimal precision and mainframe data formats	Dedicated tests for precision and for converting copybooks and packed fields; reconcile totals between systems before each cutover.
Batch processes and nightly windows	Map the schedule and job dependencies; rehearse the full batch cycle in the new environment before moving it.
Data migration	Repeatable loads, synchronization during coexistence, reconciliations, and a tested rollback plan.

Risk	How to mitigate it
Parallel run	An agreed parallel-run period with automated result comparison and exit criteria in writing.
Knowledge loss during the project	Recorded knowledge-transfer sessions and living documentation; involve the people who maintain the system from day one.
Compliance and audit	Change traceability, test evidence, and business sign-off at every cutover.

6. Readiness checklist

- ☐ You have access to the complete source code, including copybooks and JCL.
- ☐ There is an inventory of interfaces, files and batch processes, with their schedule.
- ☐ The people who know the system have time allocated to the project.
- ☐ Representative (anonymized) test data and an environment to compare results are available.
- ☐ Equivalence criteria are defined: which differences are acceptable and which are not.
- ☐ Operations, risk and audit know the coexistence and rollback plan.
- ☐ A business sponsor prioritizes the capabilities to modernize.
- ☐ It is agreed what happens to licenses, infrastructure and historical data at decommission.

7. How we approach modernization

Nextsoft does not migrate legacy systems on its own: we deliver modernization together with specialized strategic partners. The partner migrates the COBOL system; Nextsoft builds the new modules, integrations, frontend and QA, and is your single point of contact throughout the project.

Next step

Take the free legacy self-assessment: 10 questions on technology, operations, security and business, with the result calculated in your browser. If you would like to talk through your case, email us.

Self-assessment: nextsoftcorp.com/en/legacy-self-assessment

Email: contacto@nextsoftcorp.com

Sources (accessed September 2026)

GAO-19-471, Information Technology: Agencies Need to Develop Modernization Plans for Critical Legacy Systems (2019): <https://www.gao.gov/products/gao-19-471>

Wikipedia, COBOL (current standard, mainframe use, 2020 programmer shortage): <https://en.wikipedia.org/wiki/COBOL>

IBM, What is COBOL?: <https://www.ibm.com/think/topics/cobol>

Microsoft Azure Architecture Center, Strangler Fig pattern: <https://learn.microsoft.com/en-us/azure/architecture/patterns/strangler-fig>

AWS Prescriptive Guidance, migration strategies: <https://docs.aws.amazon.com/prescriptive-guidance/latest/large-migration-guide/migration-strategies.html>