INVOKE PUBLIC SECTOR

DAFITC 2026

AI-Driven Modernization of Legacy Mainframe Systems

A case study from Air Force Sustainment Command

Jorge Rodriguez
Director, Product Delivery & Strategic Growth
jorge.rodriguez@invokeinc.com
305.519.0878

Sam Hakakian
AI Strategy & Solutions Lead
sam.hakakian@invokeinc.com
410.294.2750

© Invoke Public Sector USA LLC

WHO WE ARE

From commercial RPA to a government-wide vehicle

Invoke Public Sector is a non-traditional small business delivering intelligent automation and agentic AI for federal missions. We are the AI and RPA Partner to the USAF Center of Excellence, and we build the government capability to sustain what we field.

THE PATH

01
Commercial RPA
Founded in 2019 automating the highest-burden processes for commercial clients.

→

02
Invited into the Air Force
An Air Force office asked us to bring the same practice to government workflows.

→

03
SBIR Phase I
Open topic award in robotic process automation. Concept proven against real mission work.

→

04
SBIR Phase II
Scaled the prototype into fielded capability in the hands of operational users.

→

05
Phase III and IDIQ
A government-wide single-award vehicle for Intelligent Automation-as-a-Service.

FA877124D0003
Contract number

$47M
Ceiling value

Sep 2024 to 2029
Period of performance

Federal Government Wide
Sole-source task orders to any federal agency, no new competition.

© Invoke Public Sector USA LLC

CUSTOMERS

Who We Work With

Seven years of continuous federal delivery, concentrated in the Department of the Air Force and extending across the wider federal mission space.

- AF Business & Enterprise Systems
- AF Office of Special Investigations
- Air Force Audit Agency
- 448th Supply Chain Mgmt Wing
- AF Security Forces Center
- Air University

ALSO SUPPORTING

United States Space Force · Department of Education · Social Security Administration · U.S. Citizenship and Immigration Services · Defense Health Agency

© Invoke Public Sector USA LLC

TRACK RECORD

Proven Delivery Across the Air Force

Citizen Developer Program

USAF & USSF

6,000+ personnel trained and 1,000+ automations supported across the DAF.

Agentic Code Modernization

AF LIFE CYCLE MANAGEMENT CENTER

Multi-agent platform refactoring COBOL, C, and JCL. 10M+ SLOC modernized.

Bouncer Bot

USAF SECURITY FORCES

Automated CJIS visitor vetting, roughly 70% faster, deployed at 12+ installations.

Transcript Automation

COMMUNITY COLLEGE OF THE AIR FORCE

10,000+ records processed in under 60 minutes into SAP ERP.

Criminal Justice Information Management

AF OFFICE OF SPECIAL INVESTIGATIONS

NICS case management with full business rules applied, roughly 90% faster case routing.

Agentic Cyber Suite

IRAD/SOCOM

OCO, DCO, operator training, and CI/CD review pipelines.

PARTNERS AND PLATFORMS

UiPath PLATINUM Partner · UiPath Fast Track Partner · Ideagen DevonWay · databricks · Power Automate · appian

© Invoke Public Sector USA LLC

THE NEED

The bench is retiring faster than the code

Air Force mission systems written forty and fifty years ago in COBOL, CA IDEAL, and JCL still carry the work. The developers who can safely read and change them are leaving the workforce faster than they are being replaced.

01

The talent pool is shrinking

Mainframe COBOL developers are aging out and very few are entering. Every year the number of people who can touch these systems without breaking them gets smaller.

02

The knowledge left with them

The business rules were written decades ago. They now exist only as behavior in production, not as anything a new team can sit down and read.

03

Brittle systems resist change

Decades of patches on patches. Shared copybooks and cross program calls mean the module you scoped is never the module you actually have to change.

04

Parity cannot be shown

Without a trusted behavioral baseline, nobody can demonstrate the replacement does what the original did. Programs stall at the approval gate, not at the keyboard.

© Invoke Public Sector USA LLC

THE QUESTION

Modernization means three different things

Every program office says it wants to modernize. What that actually buys depends entirely on how deep you go, and most of the market stops at the first tier.

01

Wrap it

Put a web front end on the mainframe and call it modern.

- The old code still runs underneath, unchanged.
- Every constraint, workaround, and defect survives intact.
- You now maintain two systems instead of one.
- The COBOL bench problem is not solved, only hidden.

02

Translate it

Hand each file to a model and ask for the same program in a modern language.

- Compiles clean, and the behavior quietly does not match.
- Forty years of tech debt gets carried across faithfully.
- Dead code, unused libraries, and old workarounds all make the trip.
- You end up with a forty year old system written in Java.

03

Rebuild it

Recover what the system is required to do, then build that.

- Decompose to business rules and requirements.
- Separate deliberate design from workaround, and drop the workarounds.
- Reorganise into a modern architecture, thin client and browser based.
- Prove the behavior survived against the system it replaces.

Only the third one retires the legacy system. The first two keep it alive with a nicer interface.

© Invoke Public Sector USA LLC

THE APPROACH

How you rebuild without breaking the mission

You cannot skip to the rebuild. Everything depends on first recovering what the system is actually required to do, which nobody has written down for forty years.

01

Understand

Parse every program and build the call and data graph. Read what is there before inferring anything, including the languages nobody on the team recognises.

→

02

Recover intent

Reconstruct why the code was written that way, from the source, the documentation, and the requirements of its era. That is what separates real design from a workaround.

→

03

Rebuild forward

Decompose to business rules, then build a modern architecture from them. Tech debt, dead code, and unused libraries do not make the journey.

→

04

Prove it

Compile it, run it, and compare against the executed legacy system. A deterministic comparator issues the verdict, and humans hold every acceptance gate.

WHY STEP 02 IS THE ONE PEOPLE SKIP

One legacy program we handled ran 3,000 lines to fetch the current date and time, once, at the start of the pipeline. Every record in that run got the same timestamp, and a later filter depended on that. Ask a model to refactor it and you get a correct one-line function called per record. Different timestamps, filter returns nothing, system quietly broken. The COBOL was not badly written. They simply had no library for it.

© Invoke Public Sector USA LLC

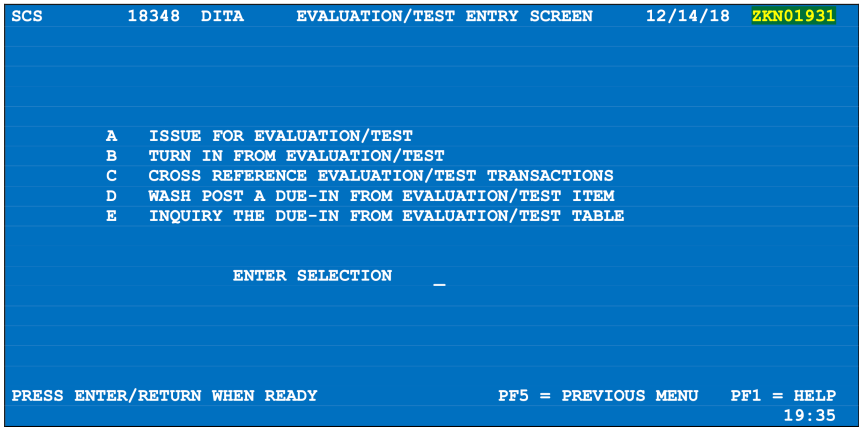
THE SYSTEM

What we walked into

SCS is a mainframe supply system supporting Air Force Sustainment Command. Its users work entirely through 3270 terminal panels, and its fielded behavior is the only authoritative specification of what it does.

THE ESTATE AS FOUND

- COBOL, CA IDEAL, and JCL in a single estate, with control flow that crosses all three.
- 3270 panels are the entire user interface. Panel identifiers, not file names, are the system's working vocabulary.
- Existing documentation described an intended system. The fielded system had drifted away from it.
- No executable test suite and no written behavioral baseline to modernize against.



SCS panel ZKN01931 as fielded

Step one was not writing Java. It was building a machine readable model of a system whose only complete description was itself.

© Invoke Public Sector USA LLC

THE PLATFORM

Agent teams do the work

Those four steps are not manual. Hierarchical agent teams run them over one shared model of the estate, at a scale no human review cycle could match, and the government holds every acceptance decision.

Understanding

GROUNDS THE WORK

Deterministic census of the estate first. Parse every program, build the call and data graph, and record what each unit touches. Nothing is inferred that can simply be read.

Delivery

BUILDS IT

An orchestrator decomposes the objective into bounded goals and hands them to architecture, frontend, service, and database agents working in isolated workspaces.

Assurance

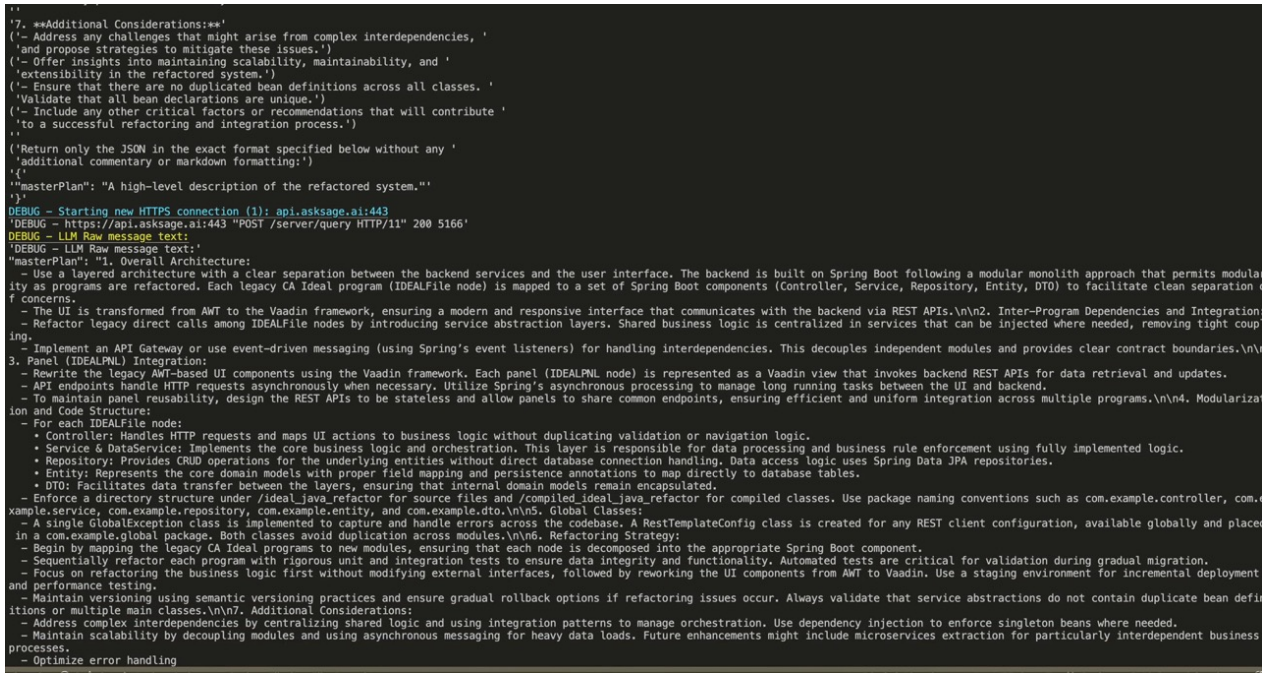
CHALLENGES IT

A separate judging pass compiles, runs, and compares the result against the legacy system. The agent that wrote the code is not the agent that grades it.

© Invoke Public Sector USA LLC

DEMONSTRATION

Agent orchestration in action: plan to code



WHAT TO WATCH FOR

- The orchestrator writes an explicit master plan before a single line of target code is generated.
- The plan names the target architecture, the service boundaries, and the panel by panel migration path.
- Legacy CA IDEAL programs and panels are addressed as nodes, which is what makes the decomposition tractable.
- Every step persists. The run can be inspected afterward without asking the agent what it did.

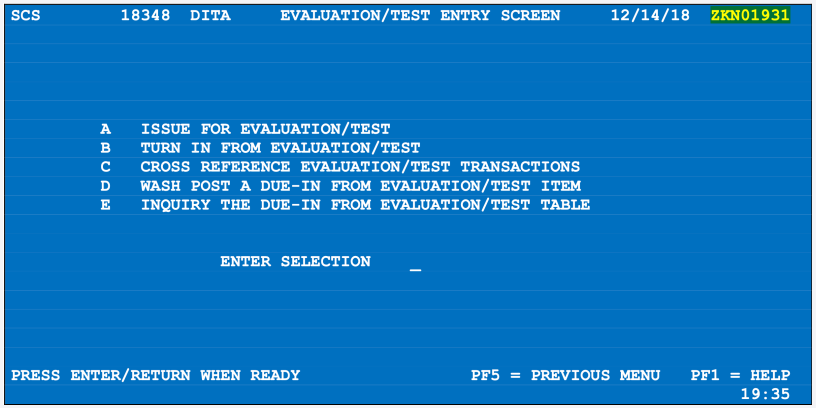
© Invoke Public Sector USA LLC

PARITY

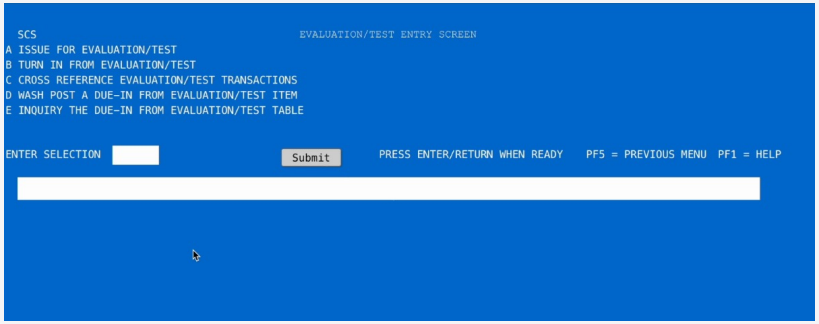
One to one first, then better

Parity is the acceptance test we can defend in front of a program office. Match the behavior exactly first, prove it, and only then move the interface forward from a keyboard-driven green screen to a browser.

LEGACY · 3270 TERMINAL



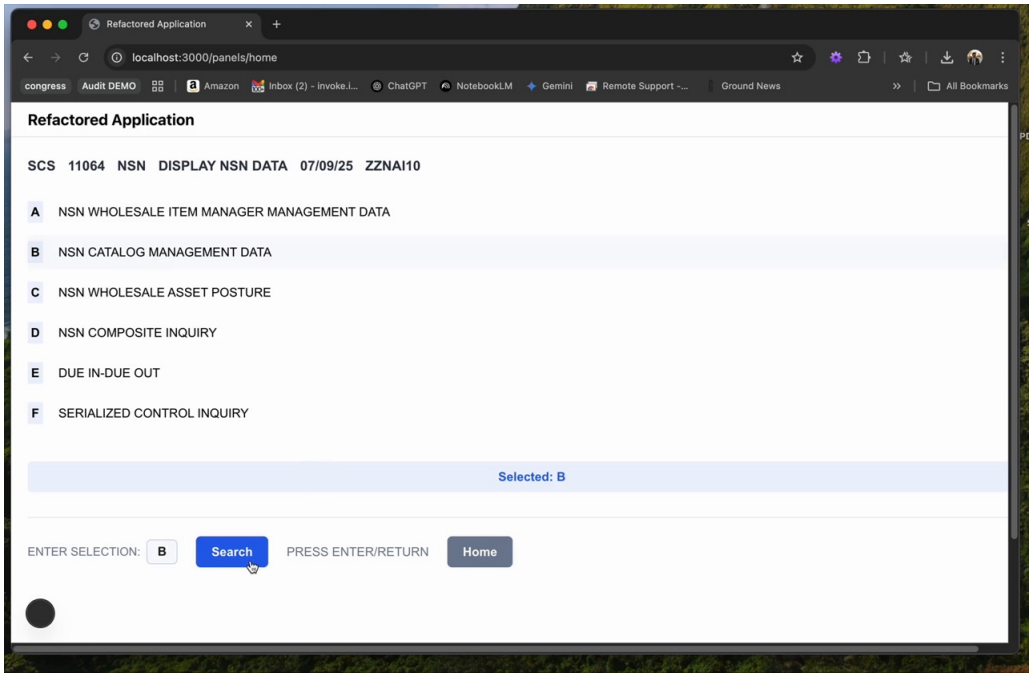
GENERATED · JAVA REPLICA



© Invoke Public Sector USA LLC

DEMONSTRATION

The modernized application



FROM PANELS TO A BROWSER

- The same panel identifiers survive into the modern application, so operator knowledge and written procedure still apply.
- Screen flow is preserved. A user who knows the sequence on the terminal knows the sequence here, which is what keeps retraining cost down.
- Underneath it is a Java service layer with a real data model, not a terminal emulator dressed up as a web page.
- Once parity is established, the interface is free to improve without putting the behavioral claim at risk.

© Invoke Public Sector USA LLC

INVOKE PUBLIC SECTOR

Thank You

Jorge Rodriguez
Director, Product Delivery & Strategic Growth
jorge.rodriguez@invokeinc.com
305.519.0878

Sam Hakakian
AI Strategy & Solutions Lead
sam.hakakian@invokeinc.com
410.294.2750

Capability Brief

Scan for the full company brief

invokepublic.com/companybrief.pdf

© Invoke Public Sector USA LLC