INGEN

Agentic Doesn't Mean Autonomous

An OpenEdge Modernization Story

PUG Americas 2026 · Jason Halbig · InGen Technologies, Inc.

INGEN

> The goal isn't to make AI better at writing ABL.
> It's to make AI **prove that it understands the ABL** before we let it change anything.

Trust isn't granted to AI. It's earned by the architecture.

Context

the evidence, source, dependencies and OpenEdge knowledge the model needs, so it doesn't guess

Control

bounded roles, deterministic steps, gates, repair limits, human approval

Confidence

hashes, provenance, compile, tests, PASOE: proof you can inspect

OpenEdge earned decades of trust by operating inside understood boundaries. AI has to earn it the same way.

About us



- **Jason Halbig** — Chief Innovation Officer, 8 years with OpenEdge
- **InGen Technologies, Inc.** — Make OpenEdge easier to run, change, and grow.
- Why we built this: customers keep asking whether AI can modernize their ABL, and "yes, trust us" is not an acceptable answer

LEADERSHIP



Mike Halbig
Founder & CEO



John Harlow
Chief Strategy Officer



Sean Garguilo
Chief Revenue Officer

"Can't AI just rewrite this?"

The hard part of modernizing ABL is **not** turning a `.p` into a `.cls`.

It's the behavior nobody wrote down:

- include files and preprocessor
- shared variables, persistent procedures
- `RUN VALUE(...)` dynamic calls
- transaction scope and rollback
- lock modes and "missing is fine" lookups
- temp-tables and ProDataSets as contracts
- UI assumptions baked into business logic
- side effects across domains

🖐 **Show of hands: who has been asked this question in the last year?**

Renaming `order.p` to `Order.cls` is not modernization

```
UI -> orderEntry.w -> order.p -> pricing.p -> inventory.p -> database
```

A model can translate that chain into classes in minutes. You now have:

- the same coupling, in a new file type
- no evidence the behavior is unchanged
- a diff too large for anyone to review honestly

Translation is cheap. Confidence is expensive.

INGEN

What we'll cover

Understand

repository · dependencies

behavior · risk

Decide

choose the seam · cited

guidance

human approval

Change

service · WebHandler

bounded generation

Prove

compile · characterization

PASOE · HTTP contract

Plus: a **live demo**, **what the AI got wrong**, and what this does **not** do yet.

The client: ForgeWorks



A synthetic field-service application — work orders, equipment, parts and service contracts.

- procedural `.p` + `.i` includes, CHUI entry screen
- temp-tables and a ProDataSet
- relational lookups, multi-table writes
- `DO TRANSACTION` with rollback
- dynamic `RUN VALUE(...)`
- PASOE + WebHandlers as the target

18 source files · 14 `.p` · 2 `.i` · 960 XREF records
Compiled with a licensed **OpenEdge 12.8.12**.

What ForgeWorks is not

A small synthetic app does not prove this works on your 3-million-line system.

It has **none** of: ADM/ADM2 or SmartObjects · super procedures · session handles · `PUBLISH/SUBSCRIBE` · `DYNAMIC-FUNCTION` · shared variables across many programs · AppServer calls · deep preprocessor nesting · triggers · COM / Sonic

> **This is the pattern we're proving.** Production discovery would expand the analyzers considerably.

INGEN

The architecture, honestly

"Is this really multiple agents, or several prompts in a workflow?"

Agentic doesn't mean autonomous

The first version of this talk had a diagram with ten AI agents talking to each other.

Building it taught us something better:

> **Not every box should be an agent.**
> Use models for ambiguity. Use deterministic tools for facts.

Models for judgment. Tools for facts.

A model does this

- summarize what a procedure does, with cited lines
- rank modernization candidates
- propose an API contract and service boundary
- review a design against cited rules
- write the service and WebHandler
- repair from real compiler output

A deterministic tool does this

- inventory, hashing, snapshots
- licensed compile + XREF
- legacy characterization + DB diffs
- OpenAPI / schema validation
- citation checks against source + RAG
- PASOE deploy, HTTP contract tests
- approval bound to exact hashes

Specialist roles → what actually implements them

Conceptual role	Implemented as	Kind
Repository analyst	snapshot + inventory + XREF parsing	tool
Dependency mapper	typed graph from source + XREF (112 nodes, 381 edges)	tool
Behavior / data analyst	`assess` node, every claim cites source lines	model
Capability selection	`select` node + recorded operator decision	human
Knowledge retrieval	`retrieve` node → Progress Agentic RAG, per role	tool
API designer / reviewers	`design`, `abl_review`, `handler_review`	model
Approval	`human_review` interrupt on an exact plan hash	human
Refactor + WebHandler agent	`generate` operation (2 files only)	model
Compiler / test agent	compile_xref · service_tests · deploy · HTTP tests	tool
Repair loop	bounded `generate` retry fed real diagnostics	model

INGEN

The real LangGraph workflows

1 · Baseline `capture_source` → `verify_legacy` → `check_readiness`

2 · Assessment `assess` → `select` → `retrieve (RAG)` → `design` → `abl_review` → `handler_review`

3 · Review `human_review` *interrupt — approve / revise / reject one exact plan hash*

4 · Execution `generate` → `compile_xref` → `service_tests` → `deploy_sandbox` → `http_and_legacy_tests`

→ report on success · ↻ a failed gate routes back to generate with the real diagnostics — max 3 repairs · 4 model requests

tool **model** **human** Checkpointed and resumable, one operation journal. An unknown outcome **stops** — it never retries blindly.

Every claim carries a label

deterministic fact

Measured by a tool: compile result, XREF record, DB diff, HTTP status, hash.

model inference

Model output that passed gates — citations resolved, schema valid — but still an interpretation.

human decision

Recorded against an exact plan hash with a rationale. Append-only.

Reviewers always know **which kind of truth** they're looking at.

INGEN

Understand

What is this application, and what does it actually do?

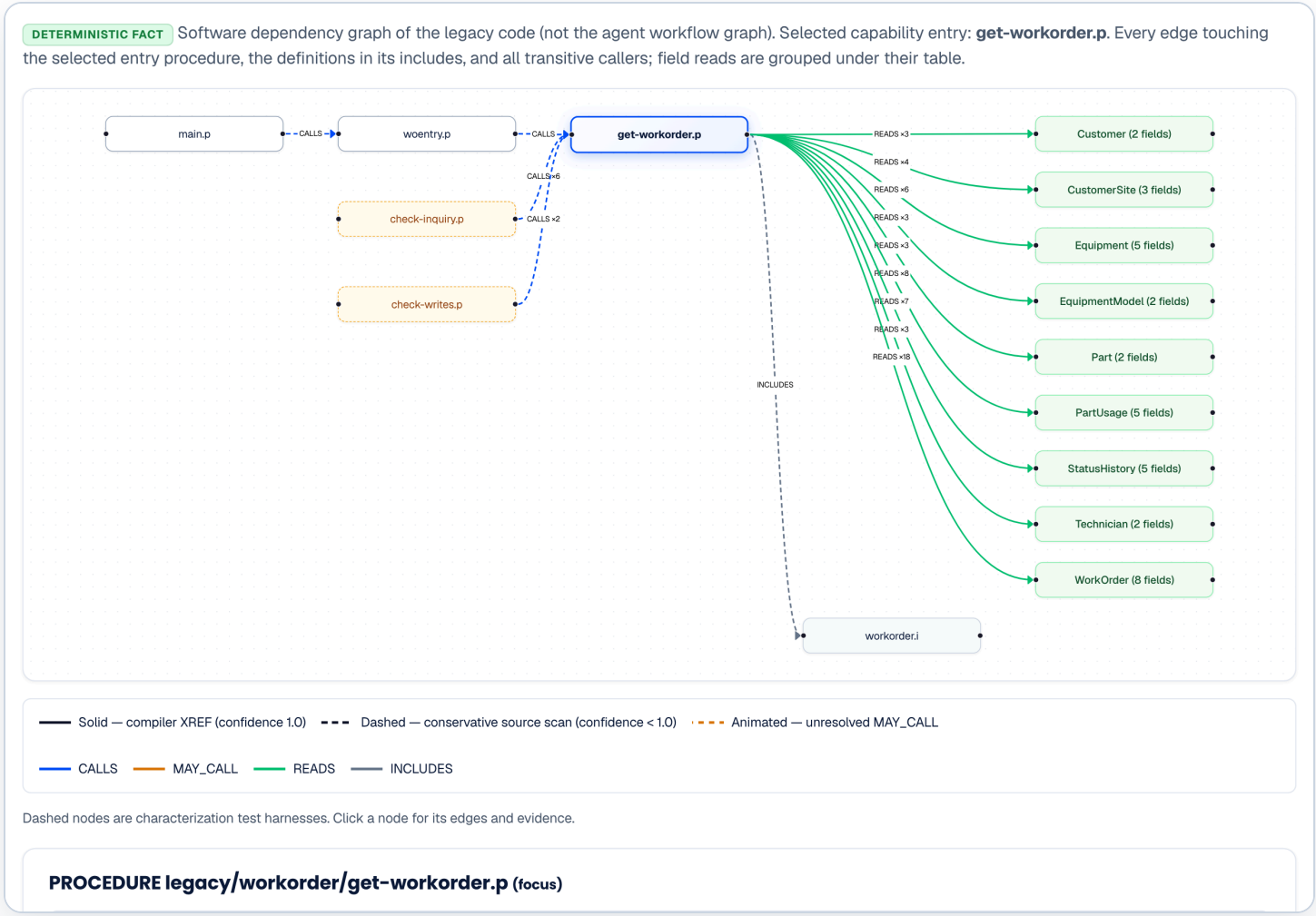
CONTEXT

Inventory and dependency graph

deterministic fact

- **18** source files
- **960** XREF records
- **112** nodes · **381** edges
- `RUN VALUE` → **MAY_CALL**, not guessed

Lexical + XREF analysis, not a full ABL parser.



Behavior, with receipts

ID	Statement	Evidence
E12	`requestedId` is an integer, must be > 0, not null	`get-workorder.p:12-15`
E14	Returns `INVALID_ID`, `NOT_FOUND`, `INVALID_REFERENCE`; output cleared on reference errors	`get-workorder.p:13-63`
E17	No persistent database writes	`get-workorder.p:3-80`
E21	Zero or more `PartUsage` per work order	`get-workorder.p:57-70`
E23	History from other work orders on the same equipment	`get-workorder.p:71-79`

model inference — every citation is **hash-checked against the exact source snapshot**. A claim without a resolvable line range is rejected.

INGEN

Observable risk, per file

Component	Writes	Transaction	Dynamic call	UI	Score
`create-workorder.p`	✔	✔	✔		9
`issue-parts.p`	✔	✔			6
`close-workorder.p`	✔	✔			6
`woentry.p` (CHUI)				✔	2
`get-workorder.p`					0

deterministic fact rubric `lexical-risk-v1` — per-file observable signals, not transitive or production risk

INGEN

Decide

Which seam first — and why not the scary one?

CONTROL

INGEN

Two procedures. Very different risk.

Work Order Inquiry — `get-workorder.p`

- read-only, `NO-LOCK`
- no transaction ownership
- no dynamic invocation
- narrow ProDataSet contract
- **a good first seam**

Create Work Order — `create-workorder.p`

- creates rows in 4 tables
- owns a `DO TRANSACTION` boundary
- `RUN VALUE(issueProcedure)`
- inventory side effects
- rollback requirements
- **not a first seam**

What the audience actually fears

```
createTransaction:
DO TRANSACTION ON ERROR UNDO, THROW:
    FIND Equipment WHERE Equipment.EquipmentId = equipmentId SHARE-LOCK NO-ERROR.
    ...
    IF CAN-FIND(WorkOrder WHERE WorkOrder.WorkOrderId = requestedId) THEN DO:
        errorCode = "DUPLICATE_ID".
        UNDO createTransaction, LEAVE createTransaction.
    END.
    CREATE WorkOrder.      ASSIGN WorkOrder.WorkOrderId = requestedId ...
    CREATE WorkOrderLine.  ASSIGN ...
    CREATE PartUsage.      ASSIGN ...
    CREATE StatusHistory.  ASSIGN ...
    RUN VALUE(issueProcedure) (requestedId, 1, openedOn, failAfterWrite, OUTPUT errorCode).
    IF errorCode <> "" THEN UNDO createTransaction, LEAVE createTransaction.
END.
```

`fixture/legacy/workorder/create-workorder.p` — `issueProcedure` is initialized to `"inventory/issue-parts.p"`, but only at runtime.

INGEN

> The graph didn't prove that Create Work Order is safe to modernize.
> It told us **why it isn't our first seam.**

We still characterized the write paths — 13 cases across create, close and parts issuance, including duplicate ID, insufficient stock and an injected failure after writes. Every rejected case left the database **unchanged** (before/after dumps compared). That evidence is ready for the next increment.

Characterize before you change

Before any code is generated, the **legacy** application must pass an independent baseline:

24
independently specified legacy cases

8
dataset comparisons

10
write pairs proving unchanged DB state

Truth data is written **independently of the model** and hash-bound. If the baseline doesn't pass, nothing downstream runs.

Progress Agentic RAG: role-specific knowledge

Role	Question it asks	Passages
design	How should the public contract relate to the legacy procedure?	5
abl-refactor	How should reads, output tables and legacy behavior be preserved?	4
webhandler	What belongs in the WebHandler, and where do DB queries belong?	5

- 4 versioned documents, each **hash-pinned** in the corpus manifest
- every citation must quote retrieved text **verbatim** — invented passage IDs are rejected
- RAG supplies knowledge; the **graph** owns state, gates and approval

What did RAG actually change?

Retrieved rule · WEB-101

"A WebHandler parses and validates the ID, calls the application service and maps its result to HTTP JSON. Business rules, database queries and transaction ownership stay outside the handler."

→

Design decision

`WebHandler → Service → get-workorder.p`

Handler owns transport + mapping only.

→

Generated code

```
oService:Fetch(iId, OUTPUT cErrorCode)
```

No FIND, no FOR EACH in the handler.

→

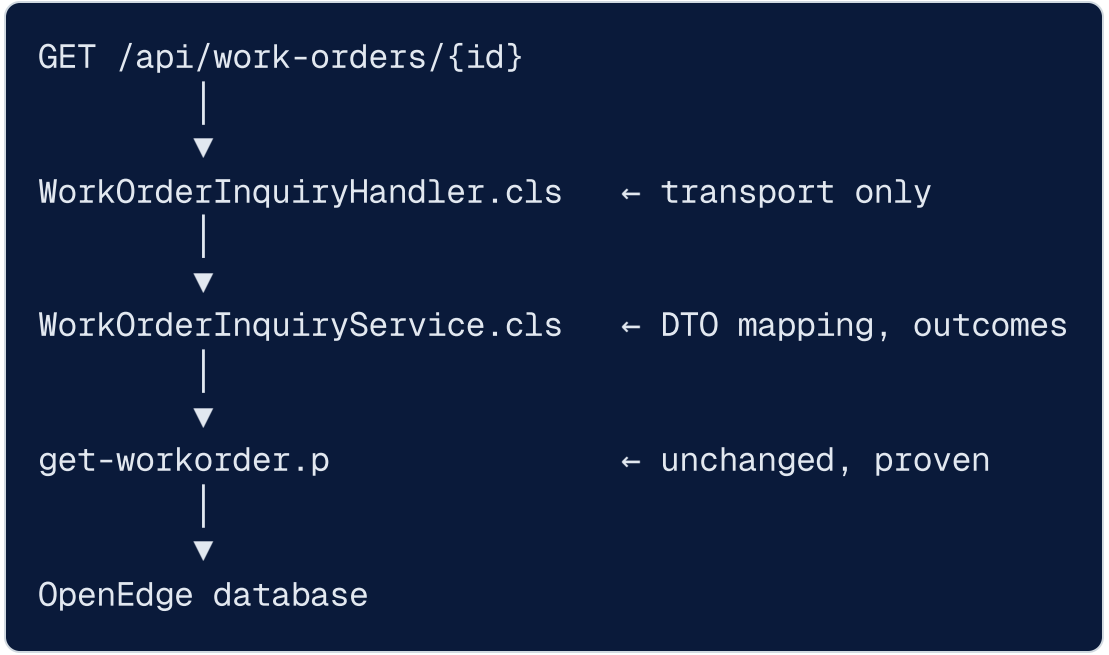
Verified by XREF

Handler: no SEARCH / CREATE / DELETE / RUN / INCLUDE.

Service: one RUN, one INCLUDE.

Knowledge → decision → code → verification. The rule is cited, applied, and then **checked by the compiler**, not by trust.

"You just wrapped the old procedure?"



Exactly. That's the strangler pattern.

- stable API boundary **without** rewriting proven behavior
- next increments move logic behind the service
- the plan explicitly says: *do not create a redundant repository class yet*

Much more defensible than asking AI to translate `get-workorder.p` wholesale into OO ABL.

Human approval is bound to an exact plan

Preserved (excerpt)

- IDs 1–2147483647; `INVALID_ID` →400, `NOT_FOUND` →404, `INVALID_REFERENCE` →sanitized 500
- ABL `FORMAT` is display, **not** a max length
- history order follows the primary index, not the CHUI's date sort
- no business-data writes; all 24 baseline cases retained

Deferred

- create / close / issue-parts, CHUI, paging, production auth

human decision

```
plan      work-order-inquiry v1
hash      sha256:b885f876a3cc…
decision  approve
```

- binds source snapshot, baseline, assessment, design and RAG evidence
- any source or prompt change **invalidates** the approval
- re-validated before every generate, repair, deploy and resume

INGEN

Change

Small, bounded, and only where the plan allows

CONTROL

The model may edit exactly two files

Model-editable

- `WorkOrderInquiryService.cls`
- `WorkOrderInquiryHandler.cls`

Hash-bound inputs it cannot change

- legacy source + `workorder.i`
- OpenAPI contract
- ABLUnit tests + runner
- characterization program + truth data
- HTTP contract tests
- PASOE configuration

A repair that returns any other file — or omits one — is **rejected**.

INGEN

The generated service

```
CLASS ForgeWorks.WorkOrderInquiryService:
    {include/workorder.i}

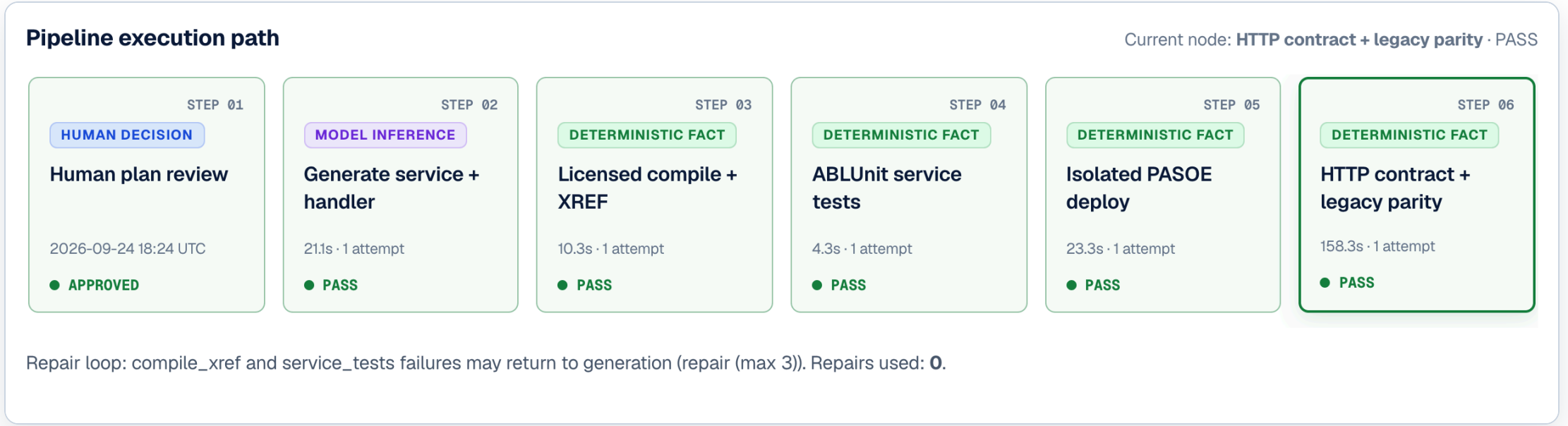
    METHOD PUBLIC JsonObject Fetch(INPUT requestedId AS INTEGER, OUTPUT errorCode AS CHARACTER):
        ...
        /* Legacy inquiry: fills temp-tables or sets errorCode */
        RUN workorder/get-workorder.p (INPUT requestedId, OUTPUT DATASET dsWorkOrder, OUTPUT errorCode).

        IF errorCode <> "" THEN
            RETURN ?.
        FIND FIRST ttWorkOrder NO-LOCK NO-ERROR.
        IF AVAILABLE ttWorkOrder THEN DO:
            oWorkOrder:Add("workOrderId", ttWorkOrder.WorkOrderId).
            oWorkOrder:Add("customerName", IF ttWorkOrder.CustomerName = ? THEN ? ELSE ttWorkOrder.CustomerName).
            ...
```

model inference 2 files, +224 lines — handler 151, service 73. Legacy procedure unchanged.

INGEN

Bounded execution



Verified run `execution-c8a865fe` · 1 model request · 0 of 3 repairs · ≈ 3½ minutes end to end.

Limits are **enforced**, not advisory: 3 repairs, 4 model requests, one operation at a time, stop on unknown outcome.

INGEN

Live demo

Understand → Decide → **Change → Prove**

INGEN

Prove

Compilation is necessary. It is nowhere near sufficient.

CONFIDENCE

The evidence chain

Gate	Result	Kind
Licensed compile + XREF architecture check	5 / 5	fact
ABLUnit service tests	4 / 4	fact
Isolated PASOE deploy, health	HTTP 200	fact
HTTP contract (service + live HTTP)	42 / 42	fact
Legacy characterization re-run after deploy	24 / 24	fact
Unchanged datasets · write pairs · implementation pairs	8 · 10 · 11	fact
Existing PASOE security smoke (untouched instance)	6 / 6	fact

This is the point of the demo — not that AI wrote ABL, but everything we required **before trusting** AI-written ABL.

The contract we verified over real HTTP

Outcome	HTTP	Owner
Successful inquiry	200	handler
Invalid path ID	400	handler
Missing / invalid credentials	401	PASOE (HTML, Basic challenge)
Missing inquiry role	403	PASOE (HTML)
Unknown work order	404	handler
Unsupported method, `Allow: GET`	405	handler
Invalid stored reference, unexpected fault	500, sanitized JSON	handler

Plus 10 edge probes: all 8 invalid-reference paths, output clearing between requests, 80-character names, unknown values as JSON `null`, history ordering.

"Who writes the tests?"

The obvious worry: *the AI writes code and tests that share the same misunderstanding.*

- characterization truth is **specified independently** and run against the **legacy** code first
- tests, contract and truth are **inputs** to generation, never outputs
- the model sees the **interface**, not the expected answers
- a test that encodes a misunderstanding fails against the legacy baseline before generation starts

INGEN

What the AI got wrong

Showing you this is the point.

✋ **Guess first:** how many execution attempts before the first fully verified run?

INGEN

Spot the bug

The schema · `forgeworks.df`

```
ADD FIELD "CustomerName" OF "Customer"
  AS character
  FORMAT "x(40)"
```

The model's API design · valid OpenAPI

```
"customerName": {
  "type": "string",
  "maxLength": 40
}
```

🖐 **What's wrong? Shout it out.**

The design mistakes

The raw model API design **passed OpenAPI syntax validation**, and still:

1. **invented** example values that weren't in the data
2. treated ABL `FORMAT` display widths as **API string-length limits**
3. proposed **JWT** authentication that wasn't configured anywhere
4. classified invalid **stored** references as **client** errors (400)

Both model reviewers missed these.

Caught by: deterministic validation against real baseline DTOs, measured PASOE behavior, and human review. Originals preserved for comparison.

The code generation mistakes

20 attempts to the first fully verified run. The 19 before it:

What the model did	What caught it
invented `OpenEdge.Web.IWebResponse` and `GetResponse`	licensed compiler
called `JsonObject:Put` (doesn't exist) and an ambiguous `Add(name, ?)`	licensed compiler
called `WebResponse:AddHeader` (doesn't exist)	licensed compiler
nested a `CLASS` inside a class file	licensed compiler
named a parameter `message` — an ABL keyword	licensed compiler
used `RUN VALUE("legacy/...")` and skipped the dataset include	licensed compiler
invented a RAG passage ID and a rule called `OEWEB-101`	citation validator

A real repair loop

```
** modern/ForgeWorks/WorkOrderInquiryService.cls
   Could not understand line 35. (196)
** ... line 36. (196)
** ... line 37. (196)
   ...
```

Cycle 16, `execution-03c7fa7e` — real licensed compiler output.

1. compile **FAIL**
2. repair-001: model gets the **actual diagnostics**, within the approved scope
3. compile **PASS** → ABLUnit **PASS** → deploy **PASS** → HTTP **42 / 42**

A repair can't change the architecture, touch a test, or exceed **3** attempts. Then a human decides.

INGEN

Honest numbers

Cost, time, and the claim we can't make yet

What it cost

Step	Measured
Assessment + design (6 model responses, first version)	441,926 input · 22,082 output tokens
Pre-implementation reviews (3 roles)	15,896 input · 2,398 output tokens
Verified generation run	**1 model request** · ≈ 3½ min wall clock
Knowledge retrieval	7 RAG requests for 3 roles
Execution attempts to first fully verified run	20

The first assessment context was **far too repetitive** — compacting XREF metadata was our biggest cost lever.

Model: gpt-4.1 · OpenEdge 12.8.12 · single-user sandbox. Most engineering time went into the gates, not the prompts.

Graph vs. one-shot prompts

Dimension	Controlled graph	One-shot 1 · bounded build	One-shot 2 · discover + build
Task	Discover, approve a plan, implement under gates	Build the specified inquiry	Discover a seam, design, then build
Output	Service + handler; legacy retained	Service + handler + route/security note	Service wrapper + route note + test plan; no handler or route
Review result	Plan approved; gates passed	Schema drift; HTTP mapping partial	Inquiry selected; API/boundary partial; incomplete route
Executable proof	Compile/XREF 5/5 · ABLUnit 4/4 · HTTP 42/42 · legacy 24/24 · PASOE 6/6	NOT RUN	NOT RUN
Run shape	Checkpointed gates; human approval; bounded repair loop	1 request · 0 repairs	1 request · 0 repairs

Takeaway: One-shot 1 tests a supplied brief; one-shot 2 adds discovery. Different scopes, **not a head-to-head**. Neither was compiled, deployed, or runtime-tested.

Static review of frozen outputs · 2026-10-01. Graph = verified inquiry run.

What this does not do yet

- not a complete ABL parser — lexical + XREF, with explicit limits
- XREF misses internal calls, dynamic targets, `PUBLISH/SUBSCRIBE`
- `RUN VALUE` targets are recorded as **MAY_CALL** uncertainty, not resolved
- risk scores are per-file signals, not transitive or production risk
- single-user: no concurrency or performance claims
- write APIs are **proposed**, not implemented
- demo authentication, not production authentication

Scaling to a real application

What would have to grow for a multi-million-line system:

Discovery

- preprocessor + nested include expansion
- persistent handles, super procedures
- shared variables across call chains
- `DYNAMIC-FUNCTION`, `PUBLISH/SUBSCRIBE`
- ADM2 / SmartObject conventions

Stays the same

- facts vs. inference vs. decision labels
- characterize before change
- approval bound to an exact plan
- model edits only planned files
- compile → test → deploy → contract gates

The **pattern** scales. The **analyzers** are where the real engineering goes.

INGEN

Why LangGraph? Why not just a coding agent?

What LangGraph gave us

- durable, checkpointed state
- interrupts for real human approval
- resume after a crash without replaying side effects
- small graphs we can reason about

Coding assistants are great for interactive work. This needed something else:

- the same steps every time
- the model cannot touch tests or config
- every step leaves a hashed receipt
- stops for a human instead of improvising

Is the generated code maintainable?

Honestly: **correct, but not what a senior ABL developer would write.**

- the handler repeats the same response-writing block **ten times**
- a human reviewer would extract a `WriteJson(status, body)` helper
- it compiles, passes 42/42, and respects the architecture — **and still deserves review**

That's why the output is a **reviewable diff and report**, not an auto-merge.

What we learned

1. **Agentic doesn't mean autonomous.**
2. **Use models for ambiguity. Use deterministic tools for facts.**
3. Choosing the seam is a bigger decision than writing the code.
4. Characterize the legacy system before the model sees the task.
5. "Schema valid" and "another model reviewed it" are not verification.
6. Most failures were **missing context**, not a weak model — exact interfaces fixed more than prompt tweaks.

Take home

- Start with **application understanding**, not code generation
- Make the model **cite** its claims, and check the citations
- Pick the **first seam** on evidence — and say why the scary one waits
- Humans approve **the plan**, not a giant diff
- Wrap proven behavior; **strangle**, don't rewrite
- Keep WebHandlers **thin** — prove it with XREF
- Treat RAG as **knowledge**, the graph as **control**

INGEN

The goal isn't to make AI better at writing ABL.

It's to make AI prove it understands the ABL before we let it change anything.

Context · Control · Confidence → Trust

Questions?

jason.halbig@ingentek.com · Repository link

INGEN

Backup

Answers to the questions we expect

Quick answers

Question	Short answer
Why the easiest procedure?	Deliberately — it's the right first seam. The assessment independently ranked it #1 and scored Create at 9
How do you resolve `RUN VALUE()`?	We don't guess. It's recorded as MAY_CALL with evidence and stays a risk
Nested includes / preprocessor?	Not solved here. XREF helps; a production analyzer needs full expansion
Persistent handles, shared vars?	None in this slice; the plan forbids introducing any
Where does approval stop the workflow?	At `human_review` — no generation without an approved exact hash
Repair loop keeps failing?	Stops after 3 repairs or 4 model requests; a human decides
Would you run this on production code?	On a copy, in an isolated sandbox, with the same gates — never against a live DB
RAG vs. docs in the prompt?	At this size, similar. At scale: role-scoped, versioned, citable, shared

What about the OpenEdge MCP Server?

It's the next step, not a replacement.

- It exposes **running** PASOE REST APIs to AI clients as MCP tools, generated from an OpenAPI spec
- It doesn't read, refactor, compile, or test ABL
- This pipeline's output is a **verified OpenAPI contract + a deployed, tested service** — exactly its input

Modernize and prove first. Then let agents call it safely.

Graph state is structured, not prose

```
source snapshot hash      approved plan hash       model-editable files
legacy baseline hash      validator hash           operation journal + receipts
inventory + XREF          gate bundle              compiler / test results
dependency graph          RAG corpus manifest      repair count · model request count
assessment (cited)        human decisions          final report
```

Each transition is traceable to **source evidence**, a **tool result**, or a **human decision** — never to a paragraph one model wrote for another.

Where to look in the repo

Topic	Location
Assessment graph	`src/modernization_graph/assessment.py`
Execution graph + review interrupt	`src/modernization_graph/workflow.py`
Analyzer and its limitations	`src/modernization_graph/tools/analysis.py`
Legacy fixture	`fixture/legacy/`
Knowledge rules (DESIGN / ABL / WEB)	`knowledge/`
Verified run report + diff	`evals/expected/modernization-report-c8a865fe-v1/`
API design and review corrections	`docs/design/inquiry-api-design.md`