The Prompt is the Product:

Field Lessons in AI-Powered Modernization - Benchmarking Claude Code, OpenAI Codex, and the Human in the Loop

Table of contents

Chapter 3:

Benchmark Phase I – Independent Program Migration

Building on our initial summer trials, we moved last month to a more structured, quantitative phase of testing. The goal shifted from exploring capabilities to benchmarking performance under controlled conditions.

For this first benchmark, we focused on high-volume, independent script migration. To understand the current state of the art, we pitted two distinct technological philosophies against each other: Autonomous Agents vs. Specialized Workflows.

The Benchmark Objective

The specific goal was identical for all participants: rewrite and test a comprehensive suite of legacy programs from the GitHub public repository cobol-examples. This repository consists of 26 distinct programs covering the full spectrum of legacy logic, including a mix of simple I/O, file handling, and database operations.

The Benchmark Lineup

The Autonomous Agents: Claude Code & OpenAI Codex

These are the "Black Box" autonomous agents designed to reason through problems in real-time. We selected Claude Code (returning from our initial assessment) and added OpenAI Codex to the roster.

We selected these two explicitly because of their undisputed market dominance. At the moment of writing, these technologies command the largest active user communities and define the industry conversation regarding AI coding. Given the massive enthusiasm and widespread adoption surrounding these ecosystems, they were the mandatory candidates for a serious benchmark. Our goal was to test if the performance matched the reputation.

The Specialized Workflow: G.Tx

To provide a comparative standard, we included G.Tx, our proprietary internal tool. Unlike the generalists, G.Tx is a Workflow Agent. It does not "improvise" a plan; it follows a strict, pre-defined modernization path (combining deterministic parsing with Generative AI).

The Surprise Factor:
Evolution of Competence

Before analyzing the metrics, one general observation stood out immediately. In our previous tests, agents would often skip complex tasks or hallucinate completion. In this benchmark, both tools attempted every single example. Neither tool "lied" about getting things done while secretly skipping files.

This shifted our measurement strategy. Since 100% "Completion Rate" is now the industry baseline rather than a differentiator, we excluded it from our final scoring and focused entirely on quality and consistency.

Measurement Methodology:
The "Three-Run" Standard

To account for the stochastic nature of Generative AI - where the same prompt can yield different results - we did not rely on a single pass. We executed the entire benchmark suite three times for each tool.

This approach allowed us to measure Standard Deviation, giving us insight into the "personality" of the models. We weren't just looking for the best run; we were looking for the most consistent one.

We evaluated success based on three specific technical metrics:

1. **Test Success Rate:** The percentage of tests that passed. We tracked the variance between runs to determine if the tool was reliable or lucky.
2. **Code Coverage:** This was our critical factor. A high pass rate is meaningless if the tests only cover 10% of the code. We required the agents to measure their own coverage, exposing whether they were rigorously testing their logic or just writing "happy path" tests.
3. **Duration:** We measured the time to completion for the full batch.

Note on Cost: While we tracked token usage, we ultimately decided to exclude Cost as a primary metric. For modernization batches of this volume, the API costs have become so marginal that they are no longer a significant decision factor for enterprise projects.

The Manual Spot-Check Protocol

Crucially, we did not rely solely on the agents' self-reported test results. To ensure the migration was genuine and not a hallucination of passing tests, we performed a manual code review and execution on three "Control Programs" after every single run:

- **Hello World:** Verified basic syntax and compilation.
- **Read File:** Verified file I/O operations and directory path handling.
- **Database Select:** Verified external dependency handling, and data retrieval.

This human review provided the "ground truth" confidence level, ensuring that high pass rates corresponded to actual, runnable code.

Methodology:

4-Phase Prompt Architecture

To eliminate variables and ensure a fair comparison, we designed a strict 4-Phase Prompt Architecture. This was not a free-form conversation; we acted as the "Modernization Expert," feeding the agents precise, role-based instructions. The agents were required to maintain state by writing to a markdown file on the file system, tracking their own progress from analysis to verification.

Phase 1: Planning and Tracking

We did not simply ask the agents to "convert code." We required them to first act as auditors. The agent had to scan the directory, identify every .cob script and .cpy copybook, and map their dependencies. Crucially, they were required to generate a conversion matrix - a markdown table tracking the status of every file. This forced the model to acknowledge the scope before writing a single line of Python.

Phase 2: Strict Implementation Standards

The conversion phase had strict rules. We required modern Python standards (version 3.11+). We also included a conditional directive regarding environment: "Only use Docker if external dependencies are required." This instruction acted as a specific hint, explicitly authorizing the agent to spin up a Docker environment if - and only if - it detected a valid external dependency like a database.

Phase 3: Automated Test Generation

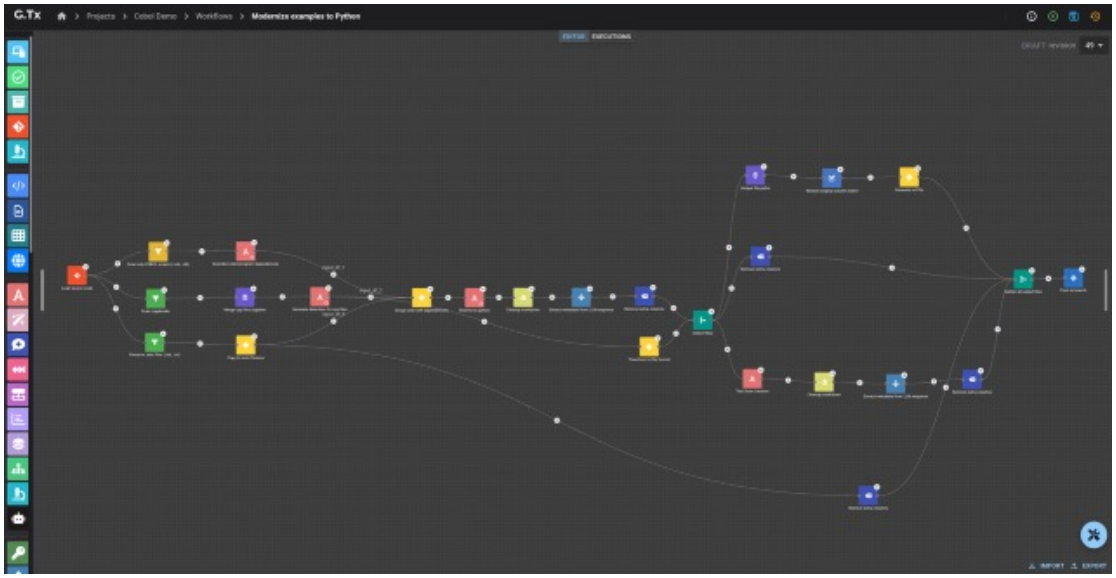
We mandated a strict verification protocol: for every new Python script created, the agent was required to generate a corresponding test file. The goal was to ensure that the generated code was functional and internally consistent. By forcing the agent to write passing tests for its own work, we filtered out code that looked syntactically correct but failed to execute.

Phase 4: Truth Serum Verification

Finally, the agent was required to run the tests itself and generate a verification report. This report had to list total tests, pass/fail counts, and coverage percentages. This phase allowed us to compare what the agent claimed happened versus the actual terminal output.

Note on G.Tx Workflow Alignment

While Claude Code and OpenAI Codex received these instructions as conversational prompts, G.Tx executed these exact requirements through a designed workflow.



G.Tx is a flexible process orchestration engine that allows us to model any modernization path. For this benchmark, we configured a specific workflow model that mirrored the exact logic of the 4-Phase Prompt Architecture. This ensured that while the execution method differed (autonomous reasoning vs. structured workflow), the steps, rules, and success criteria were identical.

Round 1: The Initial Baseline

Claude Code – The Hesitant Expert

We executed this 4-phase strategy with Claude Code using the exact prompts described above. While the tool displayed high intelligence, its behavior revealed a personality trait best described as "hesitant compliance."

Uncover the Truth About AI Modernization.

Get the full guide

grape up®

Breaking the linear