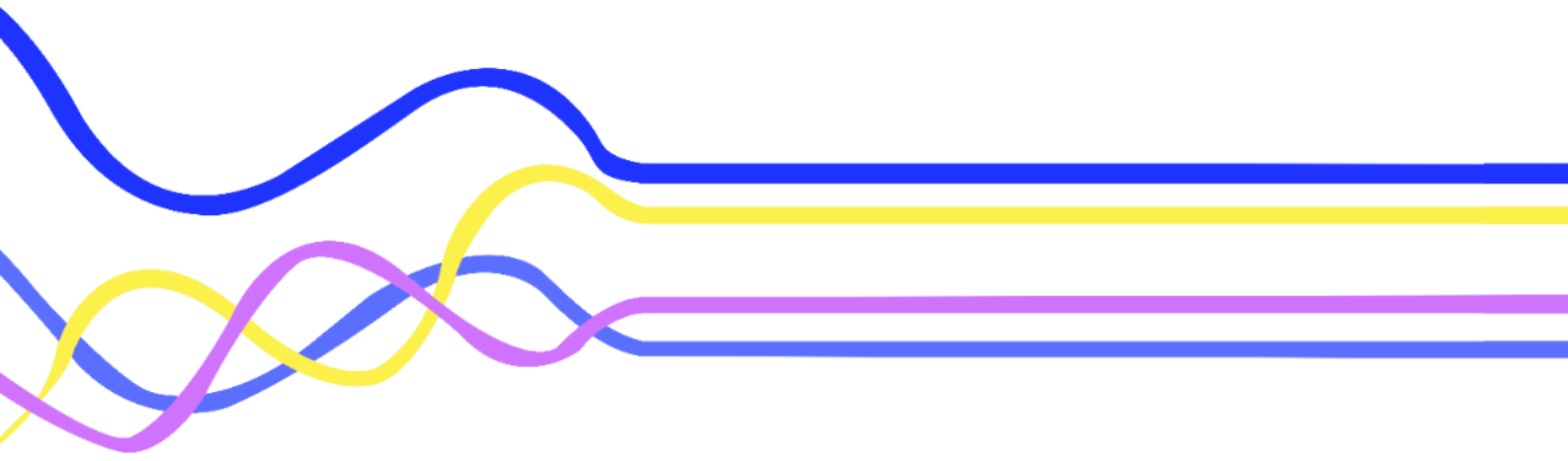
swimm

Swimm understands where Claude Code can't



Can Claude Code understand COBOL applications?

Enterprise teams evaluating mainframe modernization are asking a reasonable question: can general-purpose AI coding tools handle the understanding phase of complex COBOL application modernization?

Claude Code is a capable tool. Its use of skills, tools, and agents with powerful reasoning models enable it to accomplish developer tasks that would have been inconceivable only a few years ago. However, is it capable of solving the understanding challenge?

Understanding real COBOL applications is a different class of problem. These are programs with hundreds of thousands of lines, business rules embedded across decades of modifications, data structures built on copybooks that redefine fields in ways that resist surface-level analysis, and control flows that mix GO TO, PERFORM, and fallthrough logic in patterns no tool can shortcut.

We tested Claude Code against Swimm on real COBOL programs to find out whether an LLM-based approach can deliver that level of understanding. The results were definitive.

	Claude Code	Swimm
Coverage	Missed up to 76% of the code	100%
Accuracy	Errors in every category	Zero errors
Consistency	Different results on every run	Identical every time

What we tested and how

We selected two real CMS Medicare COBOL programs from the public domain: HOSPR210, the Hospice PPS Pricer at 4,692 lines, and OPPSCAL, the Outpatient PPS Pricer at 18,835 lines. Both are available for download from CMS.gov — anyone can run this test and verify the results independently.

To isolate analysis capability rather than prompt engineering skill, we stripped all inline comments before running either tool. Real production COBOL rarely ships with explanations.

We gave Claude Code every reasonable advantage:

- Refined prompt (~800 words) specifying output format, analysis depth, and documentation priorities
- 4-pass agentic strategy: initial scan, deep analysis, cross-reference, and synthesis
- ~70 tool calls per run
- 3 independent runs to measure consistency

Swimm processed the same files through its deterministic understanding platform with no special configuration.

Coverage, accuracy, and consistency

100%
Swimm paragraph coverage on both programs

<35%
Claude Code coverage on the larger program

0%
Swimm run-to-run variance

Program	Claude Code (min)	Claude Code (max)	Swimm
HOSPR210 (4,692 lines)	69%	86%	100%
OPPSCAL (18,835 lines)	24%	35%	100%
Run-to-run variance	12% spread	42% spread	0%

Many paragraphs in Claude's output satisfied only the first dimension of coverage. They appeared, but they said nothing useful.

> **Seventy lines of payment logic produced zero extracted rules across all three runs.**

Accuracy

Claude Code produced errors across every category of analysis. Swimm produced zero accuracy errors on the same programs. The best Claude run on HOSPR210 generated 280 business rules — we audited all of them and found 39 errors across four categories.

Error type	Count	Representative example
Meaningless descriptions	20	Paragraph described as "Sum, move, zero" — names operations but omits what is summed, moved, or reset
Misattributed rules	7	One run attributed a feature to FY2016; the actual year was FY2019
Dropped conditions	11	Payment rules omit UNITS2 > 0; extracted version implies payment applies to any claim
Logic errors	1+	PTCA table search described incorrectly in all runs

The dropped conditions are the category most likely to cause production failures. A migration built on this description produces systematic overpayments on zero-unit claims — the kind of error that clears testing and surfaces in a CMS audit.

Parameter	Correct value	Claude extracted
FY2016 IP Limit	$1,288	$1,300
FY2017 IP Limit	$1,316	$1,340
FY2015 Outlier Threshold	$2,775	$2,900
FY2017 Outlier Threshold	$3,825	$3,325

These are plausible numbers. They round cleanly. A reviewer who does not have the source code open would not catch them.

Consistency

Swimm produced identical output on every run. Claude Code did not — three runs on the same code produced three different results, with no way to determine which one to trust.

> **Three runs on the same code produced between 140 and 226 rules, with no way to know which run to trust.**

The run with the most rules produced the most errors. More output is not a quality signal, and nothing in the output tells you which run to trust. There is no confidence score, no flagged uncertainty, no structural difference between a run that extracted a rule correctly and one that invented a plausible dollar amount.

Swimm produces identical output every time on the same input. When the output is a specification document that a development team will build against, determinism is a correctness requirement.

Why Claude Code hits a structural ceiling

Claude Code's gaps are not a product of a poorly constructed prompt. The prompt was approximately 800 words, refined across multiple iterations, and executed through a four-pass agentic strategy. The gaps appeared anyway — because the LLM-based approach has a structural ceiling that better instructions cannot raise.

> **A model cannot know what it has not seen.**

When a 70-line payment subsystem sits outside the paths the agent happened to follow, the output produces no warning, no gap marker, no indication that anything is missing. It looks complete because the model has no way to represent its own blind spots.

"Validate the output" fails in two ways:

1. You cannot audit for what you do not know is absent. A reviewer checking extracted rules has no indication that an entire payment subsystem was skipped.
2. When three runs return different rule counts, determining which is correct requires returning to source code directly — the manual work extraction was supposed to replace.

Larger context windows and better agent frameworks do not change this. Deterministic reproducibility is not a capability threshold that a more powerful model crosses. It is an architectural property that probabilistic generation cannot provide by construction.

How Swimm delivers understanding

Swimm's Application Understanding Platform delivers accurate, complete understanding of legacy applications, and wraps it in a platform that manages the workflows modernization teams need.

The platform's deterministic engine parses uncompiled source code and constructs a complete structural graph — call chains, dependencies, data transformations, variable reassignments — at unlimited depth, before any AI touches it. AI translates what the deterministic layer already knows into business language; it does not discover structure or infer it.

The result is a governed, shared understanding that serves both human teams and AI tools:

- **Application Map** for navigating from business domains to processes to business rules
- **Business Rule Extraction** that produces structured, source-linked rules in plain English
- **Flow and dependency analysis** that traces execution paths at unlimited depth
- **Glossary** that converts cryptic code names to business terms across all outputs
- **Collections** that package understanding into goal-oriented deliverables
- **MCP integration** that exposes curated understanding to AI coding assistants

> **Better LLMs make this foundation more readable. They do not make it less necessary.**

swimm

Swimm Application Understanding Platform

See Swimm in action