

Same modernization, three setups: what the tester checks decides whether it helps

Claude Code ported a decades-old Fortran library to Go three times in each of three setups: alone, with a tester checking the manual, and with a tester checking equivalence with the original. Every port was judged on the same hidden checks.

Case study. Runs of 17 October 2026. An experiment by our team, 3 runs per setup.

In short

- Our team gave Claude Code one modernization in three setups: port a decades-old numerical Fortran library, unmodified, to Go behind a fixed HTTP interface, keeping the original's numerical behaviour.
- **A** worked alone. **B** implemented only, while a tester ran checks written from the library's user guide. **B'** implemented only, while a tester ran checks against the original's recorded answers.
- Every port was judged on the same 109 hidden checks against the original's answers: A passed 104, B 81, B' 106.
- B' cost \$10.04 a run, 41% less than A at \$16.92.
- What the tester checks decided whether the loop helped or hurt. Three runs per setup on one codebase: the limits are listed below.

The question

In a modernization, the job is to keep what the old system does. Does Claude Code with an independent tester do a better or cheaper port than Claude Code on its own, when every port is judged on the same hidden checks? And does it matter what the tester checks?

The setup

Item	Value
The task	Port a decades-old numerical Fortran library, unmodified, to Go behind a fixed HTTP interface, keeping the original's numerical behaviour. Every run got the same ticket, sources, user guide and interface, and never saw a check.
The model	Claude Code on Claude Opus 5.5, high effort, in every setup.
Setup A: alone	Claude Code working as it normally would. It writes and runs its own tests.
Setup B: tester on the manual	Claude Code implementing only. A tester ran 100 checks written from the library's user guide and returned only the names of the failures, for up to 4 rounds.
Setup B': tester on the original	The same as B, but the tester ran 106 equivalence checks against the original's recorded answers, half of a set of 215.
The judge	The other 109 equivalence checks, hidden from every setup.
The tester	A script that ran the checks and returned the names of the failures. It was not OneDroid Argus: our product's route could not reach the experiment's local setup.
Runs	3 per setup, on 17 October 2026.
Cost	Claude Code's own <code>total_cost_usd</code> per run. Our own count from the transcripts agrees within 0.11%.

The result

Mean per run	A: alone	B: tester on the manual	B': tester on the original
Hidden checks passed, of 109	104	81	106
The same without 5 ambiguous checks, of 104	101	78	101
Checks from the user guide, of 100	90	95 (its tester ran these)	90
Cost per run	\$16.92	\$15.19	\$10.04 (−41%)
Tester rounds used	none	4 of 4	1
The coding agent's own test runs / subagents	13 / 3	0 / 0	0 / 0

Same modernization, three setups.

A decades-old Fortran library ported to Go. Opus 5.5, 3 runs per setup. Judged on 109 hidden checks against the original's answers. No setup saw them.

Hidden checks passed (of 109)

Claude Code alone, testing itself

104

+ a tester checking against the manual

81

+ a tester checking "behave like the original"

106

Cost per run

Claude Code alone

\$16.92

+ tester on the manual

\$15.19

+ tester on the original

\$10.04 (-41%)

What the tester checks decides whether the loop helps or hurts.

Means of 3 runs, one codebase. Cost = Claude Code's own total_cost_usd.

5 ambiguous checks explain the +2. The tester's cost of writing its checks is not included.

OneDroid Argus · onedroid.ai/argus

The same numbers as a chart. Means of 3 runs per setup.

What the numbers say

- **What the tester checks decides whether the loop helps or hurts.** B's tester counted every place where the original and its manual differ as a failure, so B changed the port to follow the manual. All three B runs changed one numerical detail that the original handles approximately, which cost 22 to 25 of the 109 hidden checks. B's score on the manual's checks rose while the port drifted from the original.
- **When the tester checks "behave like the original", there is nothing to "fix".** B' stayed faithful and matched A exactly on the unambiguous hidden checks: 101 of 104.
- **B' cost 41% less than A,** because a coding agent that only implements runs no tests and no subagents. Its tester confirmed the first build, and no second round was needed.

What this experiment did not cover

Not covered	Why it matters
More than 3 runs, or another codebase	One library, 3 runs per setup: means, not a distribution.
OneDroid Argus as the tester	The tester was a script. The result is about where the checks point, not about our product.
A repair loop	B's tester passed the first build, so this does not show that iterating on equivalence repairs a port.
An ambiguous rule	One interface rule could be read two ways. It affects 5 hidden checks and explains B's 2-check lead over A.
One guard in the B and B' setup	A block on compound shell commands didn't work in any B or B' run: it failed open because a helper tool was missing.
Identical rules and instructions	Both tester setups also ran with our agent rules and an instruction to implement only. The agent working alone had neither.
The cost of writing the checks	The tester's checks came from earlier work on the same library and can be reused. Writing them is not in the cost per run.

What we take from it

- Point the tester at the behaviour you must keep. In a modernization, that is the old system's answers, not its manual.
- A tester checking the wrong thing is worse than none: setup B scored below Claude Code alone.
- Next: the same three setups where the first build fails some equivalence checks, so the loop has to repair the port.

Claude and Claude Code are products of Anthropic, named only to identify what was used. OneDroid is not affiliated with Anthropic. OneDroid Argus is built by Providentia Worldwide. Questions: michal@onedroid.ai.