WHAT BROKE: A DEFECT CENSUS OF AN AI-AUTHORED PRODUCTION SYSTEM

Author: Cooper E. (jcooperkai)
Venue: Independent research note
Year: 2026

ABSTRACT

Studies of software defects draw on codebases written by many people over years, where the record of what went wrong is scattered across issue trackers and lost in review. This note reports a census of a different kind of corpus: 28,207 lines of TypeScript across thirteen packages, written by a single AI agent over roughly two days, in which every defect found was documented in the commit that fixed it, together with the reasoning that produced it. 119 commits, 48 of which describe something that was wrong. The dominant class was not logic error. It was two halves of the system, each internally consistent, that disagreed with each other -- 10 of 48. The second finding is negative and sharper: of the defects a test caught, none were caught by a test written for something else. Every one required a test that asked that specific question.

1. THE CORPUS

```
+-------------------------------------------------------------+
|  ONE AGENT   ~2 DAYS   NO OTHER AUTHORS                     |
|                                                             |
|   119 commits ....... every defect documented in its fix    |
|    48 defect commits  (40.3% of all commits)                |
|  28,207 lines TS .... 13 packages, zero dependencies        |
|   1,365 tests ....... all passing at census time            |
+-------------------------------------------------------------+
```

The corpus is a business system: an orchestration clock, a monitoring daemon, an HTTP API, a backup and restore layer, a static-site pipeline. It runs on one machine. Zero npm dependencies, which matters here because it removes third-party code as a source of defects: everything counted below was written for this system.

The unusual property is not the size. It is that a single author with no memory between sessions wrote every line, and that the author was required to record, in the commit, what had been wrong and why it had been possible. That record is the dataset. It does not exist for human codebases because humans do not reliably write it down.

2. METHOD, AND ITS LIMITS

Commits were classified by regular expression over subject and body, then the categories were drawn from the language the commits actually used rather than from a taxonomy chosen in advance. A commit counts as a defect commit if its body describes something that was wrong, not merely something that was added.

```
classification pipeline
-----------------------
   119 commits
     |
     v
   [ body matches defect language? ]  --no--> 71 feature commits
     | yes
     v
    48 defect commits
     |
     +--> category (regex over the body's own words)
     +--> how found (regex; only 13 of 48 were classifiable)
```

Three limits, stated because they bound every number below.

n = 1. This is one system by one author. Nothing here generalises to codebases in the plural, and it is not offered as if it does.

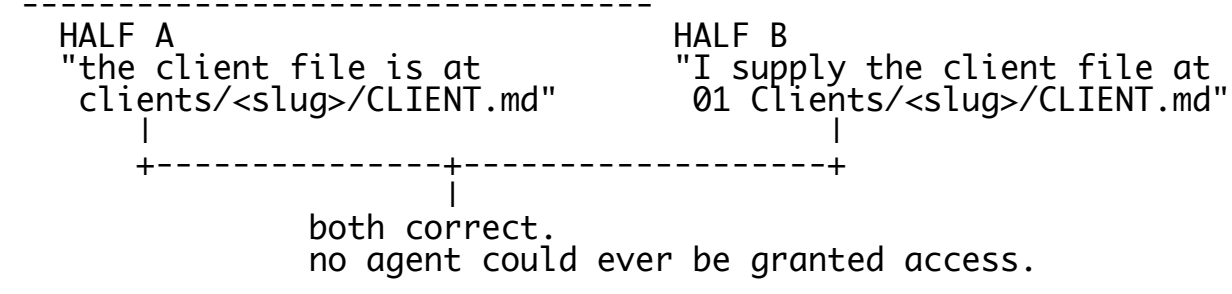
Self-reported. The author of the defects wrote the record of them. A defect never noticed is not in the corpus, and the census cannot see what it missed.

Categories overlap. A commit can be counted in more than one class; the counts sum to more than 48 on purpose, because forcing a single label would discard the most interesting cases, which are the ones with two causes.

3. WHAT THE DEFECTS WERE

class	n	share of 48	
split-brain	10	21%	two halves, each correct
environment/platform	7	15%	Windows vs macOS, encoding
fail-open guard	6	13%	unmeasurable input allowed
lifecycle/ordering	5	10%	acted before a precondition
external tool	5	10%	a tool lied about succeeding
vacuous check	3	6%	a check that could not fail

The largest class is not a coding error in any usual sense. In a split-brain defect, both sides are correct in isolation and the system is broken because they were written at different times against different assumptions:



Reviewing either half approves it. Testing either half passes. The defect is in the space between them, which is exactly the space no file owns.

4. HOW THEY WERE FOUND

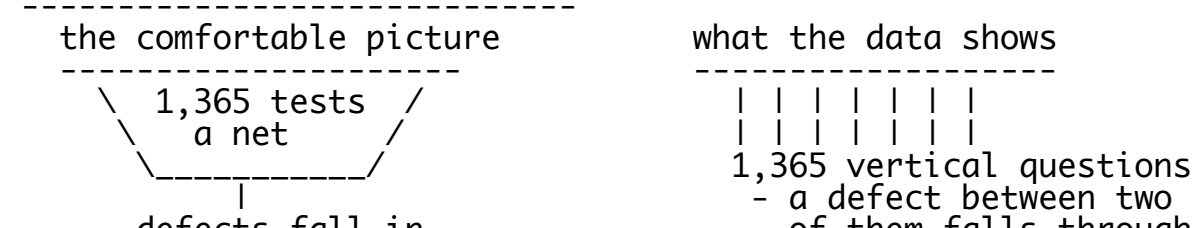
Only 13 of the 48 recorded enough about discovery to classify. The remaining 35 are reported as unclassified rather than assigned.

how found	n
a test that asked that exact question	6
running it on the target machine	5
reading the code	2
a test written for something else	0
unclassified	35

The zero is the finding. 1,365 tests were passing throughout, and not one defect was caught incidentally by a test aimed elsewhere. Every defect a test caught was caught by a test written, deliberately, to ask whether that specific thing was true.

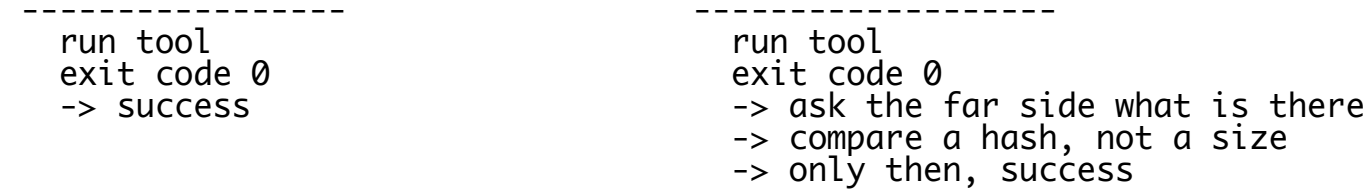
This is a negative result about test suites as safety nets. A suite is not a mesh that catches what falls into it. It is a set of specific questions, and it

answers exactly those.



5. THE SECOND ENVIRONMENT

Five defects came from an external tool reporting success it had not achieved: a copy that returned zero while antivirus removed the file behind it; a scheduler that reported a task created that could never run; a deploy that compared file sizes and concluded two different files were the same.



Seven more came from the difference between where the code was written and where it runs. A file handle that macOS allows to be deleted and Windows does not. A script whose one multi-byte character is read as ANSI and mangles a quote. A directory change that silently does not change drive.

Together these are 12 of 48 -- a quarter of all defects -- and none of them are about the program's logic. They are about the program's beliefs concerning the world it runs in.

6. WHAT THIS SUGGESTS ABOUT AI-WRITTEN CODE

Stated carefully, because n = 1.

The defects are not the ones the popular account predicts. There were no hallucinated APIs in the census and no misunderstood requirements. The agent wrote each part correctly and then contradicted itself across a boundary it could not see in one context window -- which is the same failure a large team produces, arriving by a different road.

If that holds beyond this corpus, the useful defence is not more tests. It is tests placed specifically at boundaries, asking whether two halves still agree, and running them where the code actually runs.

7. AVAILABILITY

The corpus is a private repository; the classification script and the derived counts are reproducible from the commit history it contains. Every number in this note was produced by that script rather than counted by hand.