MEASUREMENT REPORT · DATA METRICS · A DISCOVERY RECORD

Data wearing code's syntax: how one measurement anomaly became a new metric

While verifying a parser fix against Erlang/OTP, CodeDelta's numbers moved by exactly one statement — and pulling that thread led through quarter-megabyte statements in OTP's JIT, to entire GPU firmware images committed as single lines of C in NVIDIA's driver, to a question with a metric inside it: how much of your code churn is actually code? This is the record of that trail, and of the measurement it produced — shipping in CodeDelta 1.8.8.

First release 26 July 2026

ABSTRACT

Large codebases carry **data wearing code's syntax**: instruction databases, firmware images and lookup tables serialised by generators into single initializer statements. In NVIDIA's open GPU kernel modules we measured single C statements of up to **3.0MB** — and **28.4% of that tree's source lines are data, not code**. Every existing churn measure — lines or statements — mixes changes to this data into the same numbers as engineering work. CodeDelta's new **data metrics** separate them: churned statements partition exactly into code and data (CRN = code + data), element-level churn measures inside the tables ("37 of 1,647 instruction rows updated"), **working-code REWORK** reads the repair signal with table refreshes removed, and whole-tree composition states what share of a codebase is data at all. The metric's first validation produced its first finding: Erlang/OTP's striking 2026 repair regime (REWORK 22.6%) *survives the data filter unchanged* — it is genuine engineering, not table refresh. This report documents how the metric was found, defines it, and shows what it reads on real trees.

IN PLAIN TERMS

> Some “code” isn’t code. A generator takes a firmware image or an instruction set and writes it out as one enormous C statement — forty thousand lines of comma-separated values ending in a single semicolon. When that table is regenerated, every churn tool on the market reports a mountain of change, indistinguishable from a month of engineering. CodeDelta now tells the two apart: how much of your change was code, how much was data, and — inside the data — how many actual values moved. The idea arrived the way good instruments work: the tool kept flagging something odd, we followed it down, and at the bottom was a truth about codebases that nobody was measuring.

01 · The anomaly — one statement, out of two million

The trail began during release verification. A parser fix (the 64KB canonical-statement cap, documented in our limitations register) required re-deriving every published figure, and on Erlang/OTP three of seven measurement windows moved — each by *exactly one statement*, in two million statements of churn. A tool that publishes reproducible numbers has to explain a ±1, so we traced it: the moved statements lived in OTP’s JIT compiler, in vendored instruction databases where **the complete x86 instruction set — 1,647 instructions with encodings and operand rules — is one C++ array initializer of 220KB**. Its ARM64 sibling had changed during the window. Statements that size break assumptions; ours had been silently mis-tokenising them, and the fix moved the count by one.

02 · The phenomenon — five trees scanned

Was OTP an oddity? We scanned five large public trees for statements beyond 64KB:

CODEBASE	FILES WITH >64KB STATEMENTS	WHAT THEY ARE
NVIDIA open-gpu-kernel-modules	99	GSP firmware images as generated C arrays — up to 3.0MB per statement
Erlang/OTP	5	Vendored JIT instruction databases (x86, ARM64)
SapMachine (OpenJDK)	28	Vendored math tables, font tables, charset maps, FFI test data

CODEBASE	FILES WITH >64KB STATEMENTS	WHAT THEY ARE
Windows Terminal	2	Generated test tables
Chromium	0	— (the control: it generates tables at build time rather than committing them)

Four of five ship data as statements — and one of them ships *gigabytes* of it. When NVIDIA updates GPU firmware, the diff is millions of "lines of code" that no engineer wrote or will review. Any measurement that mixes this into churn is measuring the release calendar of a firmware team, not the work of a driver team.

03 · The question — "should we be parsing commas?"

Once the phenomenon was on the table, the requirement was plain: churn measurement has to differentiate data from actual code, or the data drowns the signal. Separating them decomposes into exactly three measurements:

- **The partition.** A churned statement whose initializer body holds 16 or more top-level elements is a *data statement*. Classified on the same diff trace as the headline numbers, so the identity is exact: `CRN_LLOC = code churn + data churn`. Nothing is re-defined; the headline numbers are untouched (verified byte-identical against the previous engine on real repository windows).
- **Element churn.** Inside the data statements, the comma is the unit separator: elements are hashed whitespace-invariantly and diffed, so "the table was reformatted" reads zero and "37 of 1,647 rows updated" reads 37. Nested aggregates and call arguments count as one element — a row is a row.
- **Composition.** The whole-tree standing facts: how many statements are data (`DATA_LLOC`), how many source lines sit inside them (`DATA_SLOC`), how many elements they hold (`DATA_ELEMS`).

Coverage at launch: C, C++, C#, Java, Objective-C, JavaScript, TypeScript — and Erlang's collection literals, because the discovery was made measuring Erlang/OTP and the metric ships speaking its language. (Erlang reports composition and element churn; the statement partition is a semicolon-language construct.)

04 · What it reads on real trees

NVIDIA open GPU kernel modules, calendar 2025:

```
CHURN: 117,186 statements   REWORK: 7.6%
Data churn:  566 of 117,186 churned statements are data initializers
             (elements: 2,589,790 chg / 6,866 del / 23,831 add)
Working-code:   116,620 statements | working-code REWORK: 7.5%
Composition: 584 data statements | 470,267 SLOC inside them
             (28.4% of SLOC) | 7,022,454 elements
```

Five hundred and sixty-six statements were hiding **2.6 million element changes** — the year's firmware refreshes, now measured and fenced off from the engineering numbers. And the composition line states a fact about this tree that we have not seen published anywhere: more than a quarter of its source lines are data.

Erlang/OTP, 2026 first half: our companion report found OTP in its deepest repair regime of the decade — REWORK 22.6%, nearly one churned statement in four an in-place edit. The first question a sceptic should ask is: *is that just table refresh?* The data metrics answer it: 48 of 133,428 churned statements were data; **working-code REWORK: 22.6% — unchanged**. The finding survives its own audit. That is what a second, orthogonal measurement is for.

05 · Why this matters to a reader with a codebase

- **Your churn numbers stop lying.** Effort, risk and velocity read differently when a 30,000-line diff is one regenerated table plus twelve code statements — and the report now says exactly that.
- **Review goes where the meaning is.** The twelve statements get the reviewer; the 29,988 elements get a checksum.
- **A supply-chain signal nothing else provides.** A data-churn spike in vendored files *is* an event: firmware swapped, models updated, certificates rotated — committed as arrays, invisible to line counters and licence scanners alike.
- **Metrics that cannot be gamed by regeneration.** A team measured on churn gets no credit for re-emitting a table; working-code churn is the number that survives.

06 · Availability and reproduction

Data metrics ship in **CodeDelta 1.8.8**: terminal summary lines, six churn columns and three composition columns in the CSV (name-addressed), and an HTML report row — alongside the existing metrics, never altering them. The NVIDIA window above is public history: `NVIDIA/open-gpu-kernel-modules`, `9d0b0414..2ccbad25`, one command (`codedelta --git <old>..<new>`). The OTP windows are in the companion report. Method papers: Paper A, Paper C, Paper D.

> **FIND OUT WHAT YOUR CODEBASE IS MADE OF**
>
> One scan states your tree's composition — how much is code, how much is data — and one window of history splits your churn the same way. **Download CodeDelta** — free and fully unlocked until 31 August 2026. If the data share surprises you, the tool has already paid for itself; it did for us.

References

1. Companion report: *What a line counter misses: six and a half years of Erlang/OTP* — paper-cloc-erlang.html
2. Paper A (this series), *Automated Source Code Churn Measurement* — paper-churn.html
3. Paper C (this series), *REP_CHURN: Replacement-Dominant Churn* — paper-repchurn.html
4. NVIDIA open GPU kernel modules — github.com/NVIDIA/open-gpu-kernel-modules
5. Erlang/OTP — github.com/erlang/otp; asmjit — asmjit.com