MEASUREMENT REPORT · ERLANG/OTP · 2020–2026

Scanning Erlang/OTP: six and a half years of a telecom-grade platform, statement by statement

Erlang/OTP — the open-source platform originally developed at Ericsson — is one of the hardest measurement subjects in open source: two million statements of Erlang, a C/C++ virtual machine with a JIT whose generated tables hold quarter-megabyte statements, grammars, resource files, and test suites built to be hostile. This is the record of measuring six and a half years of it properly: what it took, what broke — including in our own engine — and what the numbers say. cloc, the standard line counter, runs alongside throughout as the reference for what conventional tooling sees.

First release 26 July 2026

ABSTRACT

We measured seven consecutive windows of Erlang/OTP development (2020 to July 2026, defined by public commit hashes) with **CodeDelta** (codedelta.app): **2,022,499 logical statements of churn** across a platform that grew from 2.87M to 3.36M statements, at **93% stated file coverage** with the unmeasured remainder named in every run. Getting there took three things worth reporting: an Erlang parser built for the language as it actually is (comment signs inside strings and quoted atoms, EEP-64 triple-quoted strings, the `.app` / `.appup` /grammar/MIB family); a coverage discipline under which our own first scan — 55.6% of files, before Erlang support existed — was withdrawn as wrong-by-coverage and kept as an exhibit; and a fix to our own engine after OTP's JIT instruction tables exposed a defect in how statements beyond 64KB were tokenised — after which every published figure was re-derived. The headline finding: **2026 is OTP's deepest maintenance regime in the series — REWORK 22.6%, nearly one churned statement in four an in-place edit, against a 2020–25 range of 5.9–14.7%**. For reference, cloc 2.10 was run on identical

> hashes throughout: its default run completed four of the seven windows, reporting physical lines in changed files.

> **IN PLAIN TERMS**
>
> We measured what six and a half years of development actually did to the platform behind telecom infrastructure and some of the world's largest messaging systems. Doing it honestly meant admitting our first attempt measured barely half the project, building real Erlang support, and fixing our own tool when OTP's machine-generated JIT tables — single statements a quarter of a megabyte long — caught it out. The reward is a number a line counter cannot produce: in 2026, OTP's developers are editing existing code in place at nearly twice the rate of any year since 2020 — deep, hands-on repair of a mature platform, measured, not guessed.

01 · The subject

Erlang/OTP (github.com/erlang/otp) is the open-source Erlang runtime, standard library and toolchain, originally developed at Ericsson and still maintained by its OTP team. As a measurement subject it is close to worst-case by construction: a large Erlang corpus; a C/C++ virtual machine including a JIT compiler with vendored, machine-generated instruction databases; leex/yecc grammars; application resource files; SNMP MIBs; build systems; and test suites that deliberately exercise Unicode filenames and hostile archive content. The mid-2026 snapshot, as CodeDelta accounts for it:

QUANTITY	MEASURED
Files in tree	11,816
Total lines (LOC)	4,721,753
Source lines (SLOC)	3,706,979
Logical statements (LLOC)	3,355,831
File coverage (stated on every run)	93.0%
Unclassified text (counted for size, excluded from churn)	3,004 files

QUANTITY	MEASURED
Not measured — named per run (images, compiled .beam, certificates)	831 files

> *LOC counts every line; SLOC excludes blanks and comments; LLOC counts logical statements — the unit all churn figures use. Every file lands in exactly one bucket: measured, unclassified text, or named as not measured. Nothing is silently dropped.*

02 · The first scan was wrong — and coverage is why we could tell

Our first pass at OTP predated CodeDelta's Erlang support. The engine measured what it knew — the C virtual machine and documentation markup — and its own coverage line reported the verdict: **55.6% of files measured**. That number is what saved us. A churn report on 55.6% of a codebase is not a churn report on the codebase, so the audit was withdrawn and kept as a companion exhibit of the principle this report is built on: *a measurement without a stated denominator is a number, not a measurement.* Every CodeDelta run prints its file coverage, its unclassified-text tier, and the names of what it did not measure. The reference line counter, for comparison, excludes files silently unless you know to ask for its `--ignored` log — on this tree, 4,046 files set aside, 2,764 of them "language unknown".

03 · Teaching the engine Erlang

Erlang support was built against the language as OTP writes it, and validated against OTP's own sources:

- **Comments that aren't comments.** Erlang's `%` appears inside strings, quoted atoms and character literals (`$%`); since EEP-64, inside triple-quoted doc strings. A tokeniser has to know the difference — a 16-line golden file with known-by-construction truth (14 code lines, 6 comment lines) reads 14/6 in CodeDelta. The two regular expressions that define Erlang in the reference tool read 14/2, and on OTP's real `argparse.erl` misclassify code inside `-doc """…"""` blocks as comments.
- **The platform family, not just `.erl` .** Modules and headers, `.app` / `.app.src` / `.appup` / `.rel` resource files, escripts, leex/yecc grammars (`.xrl` / `.yrl`), SNMP MIBs, `Emakefile` and `rebar.config` — the files an Erlang system is actually made of.

- **Regression-guarded.** A 12-case Erlang tokeniser battery and a 50-file coverage canary run before every release, so support cannot silently rot.

With Erlang support and a full extension audit, coverage on OTP went from 55.6% to **93.0%** — and the remaining 7% is named, not hidden: images, compiled `.beam` artifacts, test certificates.

> ***Scope of Erlang support (CodeDelta v1.8.8 and later).*** *Support comprises full churn measurement across the platform family — `.erl` / `.hrl` modules and headers, application resource files (`.app` , `.app.src` , `.appup` , `.rel`), escripts, leex/yecc grammars, SNMP MIBs, `rebar.config` and `Emakefile` — with token-aware comment classification (`%` within strings, quoted atoms, `$` -character literals and EEP-64 triple-quoted strings is treated as content), CHG/DEL/ADD churn under similarity classification, REWORK, per-run coverage accounting, and data composition with element-level churn for Erlang's collection literals. Two properties follow from the language's shape rather than from any limitation of the implementation: Erlang has no C-style statement terminator — its comma separates expressions, its semicolon alternates clauses, and the full stop closes whole forms — so logical lines are taken equal to source lines (LLOC = SLOC), the same rule applied to Python, Ruby and shell; and the working-code/data churn partition is defined only for semicolon-delimited languages, so for Erlang the data metrics comprise composition and element churn.*

04 · The quarter-megabyte statements — and the bug they exposed in our engine

OTP's BEAM JIT emits native machine code through a vendored assembler library, asmjit, whose instruction databases are machine-generated: the complete x86 instruction set — 1,647 instructions, opcode encodings, operand rules — serialised as *one C++ array initializer*. A single logical statement of 220KB. Its neighbours are similar: 246KB of emitter macros in `x86emitter.h` , a 147KB ARM64 table that changed during 2026. Nobody writes statements like this; generators do — and vendored generated code is exactly what large real codebases carry.

Measuring them honestly cost us a defect of our own. CodeDelta's canonical statement buffer caps at 64KB, and two bugs lived past that cap: a change beyond the cap in a lone oversized statement went undetected, and — subtler — a brace-classification state variable froze when the buffer filled, so an oversized initializer among other statements could collapse to an empty token and its changes vanish from statement-level churn. The physical-line layer caught such changes throughout (both layers ship in every report), but the statement layer was wrong, and this codebase is what exposed it. Both defects are fixed: content beyond the cap now streams into a whitespace-invariant fingerprint appended to the canonical token, brace state stays live past the cap, and the fix was verified three ways — the reproducer battery (now must-pass), the full 61-test suite, and a byte-identical A/B on a real repository window proving sub-64KB measurement unchanged.

Then every figure in this report was re-derived with the fixed engine. Three windows moved — by exactly one statement each (2020 and 2025 up one, 2026 down one). We publish that correction rather than hide it: it is the difference between a tool vendor's marketing and a measurement you can audit.

05 · The results — seven windows, and one regime change

One command per window, `codedelta --git <old>..<new>`, whole-platform accounting. cloc 2.10 ran on identical hashes as the reference (its diff mode reports physical lines of code in changed files only; its default run aborts on the 2020, 2023 and 2025 windows — OTP's test suites contain Unicode filenames its git-archive step cannot package[1]).

WINDOW	CODEDELTA — LOGICAL STATEMENTS, WHOLE CODEBASE			CLOC (REFERENCE) — CODE LINES, CHANGED FILES ONLY		
	CHURN (LLOC)	REWORK	CODEBASE AT END (LLOC)	MODIFIED	ADDED	REMOVED
2020	722,671	5.9%	2,868,225			run failed [1]
2021	262,506	9.7%	3,065,399	13,497	74,817	50,071
2022	137,828	14.7%	3,083,293	18,185	41,259	23,820
2023	178,823	8.4%	3,193,943			run failed [1]
2024	362,543	7.1%	3,124,040	24,228	74,878	104,635
2025	224,700	9.3%	3,302,795			run failed [1]
2026 (Jan–Jul)	133,428	22.6%	3,355,831	30,456	49,119	94,011

> [1] `Failed to create tarfile of files from git` — *reproducible six runs out of six; the trigger is real repository content (a test directory named* `sftp_tar_test_data_` *, an emoji-named directory under* `erts/test/erlc_SUITE_data/src/` *). With git's* `core.quotepath` *disabled cloc completes these windows (2020: modified 9,954 / added 193,653 / removed 173,540; 2023: 12,620 / 75,643 / 39,067; 2025: 4,745 / 51,155 / 87,616 code lines). CodeDelta figures are from engine 1.8.8 including the §04 fix.*

The finding is in the REWORK column, and it needed everything above to exist — statement identity, whole-tree context, correct parsing. Expansion years (2020, 2023–25) run 5.9–9.3%; the 2022 consolidation ran 14.7%; and **2026 stands alone at 22.6% — nearly one**

churned statement in four is an in-place edit of existing code, the platform's deepest repair regime in the series, on a codebase that grew +17% across the six and a half years. A line count, however careful, does not contain this number: physical-line diffs cannot distinguish an edited statement from a reformatted one (a pure reformat of one real file produced 80 lines of "churn" in the reference tool and exactly 0 changed statements in CodeDelta), and changed-files-only diffing has no denominator to read change against.

06 · What the reference tool sees

None of this is a criticism of cloc at its own job — counting lines, quickly, in 397 languages — and its ubiquity is why it is the reference here. But teams increasingly quote line-diff numbers as *churn*, and this report is the measured case for why that substitution fails on a real platform: three of seven windows failed outright on legitimate repository content; the diff universe was changed-files-only, with no whole-codebase accounting (running it full-tree on this repo also meant watching it extract OTP's archive test data — including a file named `exploit.zip` — before stopping at an interactive prompt); Erlang comment classification was structurally wrong inside strings and doc blocks; and reformat noise counted as change. The two tools are not competitors measuring the same thing badly and well — they measure different things, and the table shows both.

07 · Reproduce every figure

Repository: `github.com/erlang/otp`, branch `master`. Windows (old..new):

```
2020:  acd5185edf4f..1143ad4184a3      2021:  1143ad4184a3..df450823fcdb
2022:  df450823fcdb..119b7ef1397e      2023:  119b7ef1397e..f2cfcafecd0f
2024:  f2cfcafecd0f..1a2083e48865      2025:  1a2083e48865..28a44634fb04
2026H: 28a44634fb04..6f8ac596cdd9

codedelta --git <old>..<new>          # CodeDelta 1.8.8, one command per window
cloc --git --diff <old> <new>         # reference, cloc 2.10
```

Method papers: Paper A (churn measurement), Paper C (REWORK/REP_CHURN), Paper D (REWORK baselines, 1,412 repo-year corpus).

> **RUN THIS ON YOUR OWN PLATFORM**

> Everything above was one command per window on public history — and it runs the same way on private code, on your machine, with nothing leaving your environment. **Download CodeDelta** — free and fully unlocked until 31 August 2026 — and measure the codebase you are responsible for: statement-level CHG/DEL/ADD, REWORK, per-file accounting, stated coverage, HTML/CSV reports, CI gates. If you have only ever counted lines, the REWORK column is the one you have never seen.

References

1. Paper A (this series), *Automated Source Code Churn Measurement* — paper-churn.html
2. Paper C (this series), *REP_CHURN: Replacement-Dominant Churn as a Process Signature of AI-Assisted Development* — paper-repchurn.html
3. Paper D (this series), *REWORK Baselines by Project Type* — paper-baselines.html
4. Erlang/OTP — erlang.org; source: github.com/erlang/otp
5. EEP-64, *Triple-Quoted Strings* — erlang.org/eeps/eep-0064
6. asmjit — asmjit.com (vendored in OTP's BEAM JIT)
7. cloc, A. Danial — github.com/AlDanial/cloc (v2.10, reference comparator throughout)