MEASUREMENT PAPER · TRUE_CHURN · AUGUST 2026

True churn: separating what developers wrote from what tooling produced

In one real release of a widely-used project, ***80.1% of all measured churn came from a single generated file****. A manager reading that project's churn figure was reading, four parts in five, the output of a dependency resolver. This paper defines the distinction, measures it on public code, and shows what a churn number means once the machine-written share is separated out.*

First release 1 August 2026 · CodeDelta engine 1.9.1

ABSTRACT

Churn — how much code changed between two points — is the number managers use to answer questions that matter: where is the risk concentrated, what is being rewritten, is anything moving that should not be. That number is unreliable in a specific and correctable way: modern repositories commit large volumes of **machine-generated** content — dependency lockfiles, minified bundles, code-generator output — whose churn is counted identically to hand-written work. We measured the public `npm/cli` release **v10.9.0 → v11.0.0**: **6,134** logical statements churned, of which **4,916 (80.1%)** came from `package-lock.json` alone — a file no developer edits. **TRUE_CHURN**, the remainder, was **1,218** statements. Both figures are correct; only one answers a manager's question. CodeDelta classifies generated content deterministically, subtotals it, and reports authored churn beside the total — nothing is excluded, and every classified file is named with the rule that matched it.

IN PLAIN TERMS

> If you ask “how much did this release change?” and the answer is six thousand statements, you might reasonably conclude a great deal happened. In this release, five thousand of those came from one file that a tool rewrote automatically when somebody upgraded a dependency. The developers wrote about twelve hundred. Both numbers are true. Only the second tells you what the team did.

01 The problem: churn counts the machine’s work as yours

A modern repository is not only source code. It carries content that is *derived* — produced mechanically from something else, and regenerated rather than edited:

- **Dependency lockfiles** — `package-lock.json`, `yarn.lock`, `Cargo.lock`, `go.sum`, `composer.lock`. A one-line version bump can rewrite thousands of lines.
- **Minified and bundled assets** — `*.min.js`, source maps: the same code, mechanically transformed.
- **Code-generator output** — protobuf stubs, ORM models, API clients, and the machine-produced tables that vendored libraries carry. These usually announce themselves: `@generated`, `DO NOT EDIT`, `Code generated by …`.

All of it is legitimately in the repository. None of it is authored in the sense a manager means when asking what changed. Counted together with real work, it does not merely add noise — at realistic volumes it *dominates*, and the resulting figure moves with dependency upgrades rather than with development.

> **The operational definition.** *Authored churn* is change made in the file where the work itself lives — by a human or an AI assistant; hand-maintained data files count. *Generated churn* is change mechanically derived from work done elsewhere. The test: **could this file be deleted and reproduced mechanically from other sources?** If yes, its churn is generated. The equivalent human test: would a reviewer ever read this diff line by line?

02 Measured: npm/cli v10.9.0 → v11.0.0

A real release of a real project, chosen because it is public, reproducible, and commits its lockfile — as most JavaScript projects do. Both trees are the published release tarballs; the command and hashes are in §05.

MEASURE	STATEMENTS	SHARE OF TOTAL
Total churn (CRN_LLOC)	6,134	100%
Generated churn	4,916	**80.1%**
TRUE_CHURN (authored)	**1,218**	**19.9%**

Per file, the concentration is starker still. 145 files changed in this release; here are the largest by churn:

FILE	CHURN	SHARE	AUTHORED?
`package-lock.json`	4,916	80.1%	generated — npm lockfile
`CHANGELOG.md`	861	14.0%	authored (release notes)
`workspaces/libnpmhook/README.md`	204	3.3%	authored
`workspaces/libnpmhook/CHANGELOG.md`	132	2.2%	authored
`.github/workflows/ci-libnpmhook.yml`	116	1.9%	authored
`workspaces/arborist/test/arborist/build-ideal-tree.js`	115	1.9%	authored (test)

One file outweighs the other 144 combined by roughly four to one. It is the file least likely to interest anyone asking what the team changed, and it is the file that dominates every unfiltered churn metric, chart and trend line drawn from this repository.

The scan also found **100 generated files** in the tree overall — 96 lockfiles across the workspaces plus four carrying `DO NOT EDIT` markers — totalling **98,715** logical statements

of generated content sitting alongside the source.

03 Why this breaks the questions managers actually ask

“Is anything unusual happening in our code?”

Rogue development — unreviewed work, changes outside the plan, activity in a component nobody was meant to touch — shows up as churn where churn was not expected. If four fifths of the signal is a dependency resolver, the unexpected activity is a small perturbation on a large noisy baseline. Worse, the noise moves: a routine `npm update` produces a churn spike indistinguishable in magnitude from a substantial piece of unplanned work.

“How much are our mission-critical files changing?”

This is a per-file question, and per-file is exactly where the distortion is worst. A safety-relevant module with 40 statements of careful change is invisible next to a lockfile with 4,916. Ranked lists of “most-changed files” — a standard risk-review artefact — are led by files nobody wrote.

“Is our AI-assisted development changing the shape of our work?”

Process signatures such as REWORK (the share of churn that edits existing statements in place) are ratios. Diluting the denominator with generated content compresses every such ratio toward the behaviour of the generator, not the team.

> **The failure is silent.** No tool reports an error; a number is produced, it looks plausible, and it is wrong for the purpose. That is the most expensive kind of measurement defect — it does not announce itself, and it survives being charted, reported and acted upon.

04 How CodeDelta separates them

Classification is deterministic — no inference, no scoring — and runs in three passes over each file:

1. **Known basenames.** The lockfile family: `package-lock.json`, `yarn.lock`, `pnpm-lock.yaml`, `Cargo.lock`, `composer.lock`, `Gemfile.lock`, `poetry.lock`, `uv.lock`, `go.sum`, `packages.lock.json`, `flake.lock`.
2. **Generated-output suffixes.** `.min.js`, `.min.css`, source maps, `.pb.go` / `.pb.cc` / `_pb2.py` and the protobuf family, `.g.cs`, `.freezed.dart`, analysis-tool output.
3. **Generator markers.** A bounded read of the first 2 KB for the notices generators stamp into their own output — `@generated`, `DO NOT EDIT`, `Code generated by`, `automatically generated`. This is the file declaring itself, in a machine-readable way, and it catches generators no list could enumerate.

Teams add their own rules through `codedelta_agents.json` without waiting for a release.

Additive by design: nothing is deducted

Generated churn stays inside every existing total. CRN_LLOC, CHG/DEL/ADD, REWORK and the longitudinal database are computed exactly as before, so historical comparisons remain valid and no published figure moves. TRUE_CHURN is a derived companion:

```
TRUE_CHURN = CRN_LLOC − CRN_GEN_LLOC
```

Generated churn is *not* a fourth churn class. A generated file's statements are classified CHG / DEL / ADD like any other file's, inside the corresponding headline columns, and subtotalled per component. From the npm/cli measurement:

	CHG	DEL	ADD	CRN
Totals (headline)	1,242	2,242	2,650	6,134
of which generated	776	1,745	2,395	4,916
authored = difference (TRUE_CHURN)	**466**	**497**	**255**	**1,218**

Every classified file is reported by name with the rule that matched it, in the report's coverage block, the CSV `GEN` row, the console and a GUI tile. A reader can therefore check the classification rather than trust it — and disagree with it on the evidence.

05 Method — reproduce it

Two public release tarballs, one command:

```
curl -L https://codeload.github.com/npm/cli/tar.gz/refs/tags/v10.9.0 | tar xz
curl -L https://codeload.github.com/npm/cli/tar.gz/refs/tags/v11.0.0  | tar xz

codedelta cli-10.9.0 cli-11.0.0 -o report.html --csv metrics.csv
```

The console prints the split directly:

```
CHG_LLOC: 1242   DEL_LLOC: 2242   ADD_LLOC: 2650   CHURN: 6134
TRUE_CHURN: 1218  (CHURN minus generated churn)
Generated files: 100 (generator marker (DO NOT EDIT) 4, npm lockfile 96) - 98715 LLO
Generated churn (subtotal): 776 CHG | 1745 DEL | 2395 ADD | 4916 CRN
                            -> authored churn: 1218 of 6134 CRN_LLOC
```

Measured with CodeDelta engine 1.9.1 (build 35) on the unmodified published tarballs. The unit is the **logical statement** (LLOC), not the physical line — see the companion paper on why that distinction matters before any of this does. Exact, repeatable diff arithmetic: no sampling, no machine learning in the measurement.

06 What to do with the number

- **Report TRUE_CHURN when the audience is people.** Total churn answers “how much did this repository change?”; TRUE_CHURN answers “how much did this team change?” Management reporting, risk review and effort discussion want the second.
- **Keep total churn for storage and integrity.** It is the complete arithmetic and it reconciles against the tree. Both are reported; neither replaces the other.
- **Rank files by authored churn.** A most-changed-files list that begins with a lockfile is telling you about your dependency policy, not your code.
- **Watch the ratio.** A project where generated churn is a rising share of the total is one where the reported figure is progressively less about the team — worth knowing before someone draws a conclusion from the trend.

> **The honest limit.** Classification is deterministic and inspectable, but it cannot read intent. A hand-maintained JSON configuration is authored; a lockfile is not; a file both generated and then hand-edited is a genuine grey case. Every classified file is named with its rule precisely so that judgement stays with the reader — and a wrong call is visible rather than buried in a total.

References

1. Paper A (this series), *Automated Source Code Churn Measurement* — paper-churn.html
2. Companion paper, *Why LLOC is what really counts* — paper-lloc.html (why the unit is the logical statement)
3. npm CLI — github.com/npm/cli, releases v10.9.0 and v11.0.0
4. CodeDelta user guide, “Authored vs generated churn” — user-guide.html