MANAGEMENT BRIEF · KEY TERMS

The CodeDelta Management Brief

Every term the reports use, in plain language — the 60-second version. One line each, no mathematics required.

Print-friendly PDF: download the brief · deeper treatment of any term: the papers

Measuring change

Churn (CRN)	The total amount of change between two versions of a codebase: everything added, deleted, or edited. The headline number.
ADD / DEL / CHG	Every changed statement lands in exactly one of three piles: **added** (new), **deleted** (removed), or **changed in place** (still there, but altered — an *inline edit*). Colours everywhere in the reports: green = added, blue = deleted, red = changed.
LLOC vs SLOC	**SLOC** counts pure physical source lines — total lines minus blanks and comments (comment lines are reported separately as COM_LOC). **LLOC** counts logical statements — what the program actually does. Reformatting a file moves lots of lines and zero statements, which is why serious change measurement uses LLOC.
Inline edit (CHG_LLOC)	A change inside an existing statement — not an addition, not a deletion. The report column is CHG_LLOC. Humans do this constantly; AI tools barely do it at all.
Coverage	The percentage of files actually measured, stated on every run — so nothing is silently skipped.

The shape of change

REWORK	The share of all churn spent editing existing code in place (CHG ÷ CRN). Established hand-maintained projects sit near 17% — one statement in six. A measured agent-built codebase: 0.19% — one in five hundred.
REP_CHURN	The mirror of REWORK: the share of churn that was replacement — deletions plus additions — rather than editing. High replacement is the signature of generate-and-regenerate development. REWORK + REP_CHURN = 1, always.
TRUE_CHURN	Churn from files your developers actually wrote. Machine-generated files — lockfiles, minified bundles, code-generator output — are subtotalled out by named rules, nothing hidden. In one real npm release, 80% of the "churn" came from a single generated file.
Data metrics on the GUI: the DATA SHARE tile	Some "code" is data wearing code's syntax — firmware arrays, generated lookup tables — where one statement can hold thousands of values. CodeDelta splits working code from data, counts change inside data *per element*, and reports a working-code REWORK with the data noise removed.
Baseline & merge gate	A baseline is an accepted snapshot of findings. The gate fails a pull request only on findings *newer* than the baseline, or on a policy breach — measurement becomes enforcement only when you say so.

How your team runs it (no coding required to read the results)

Desktop app	Point-and-click on macOS, Windows or Linux — pick two versions, get the report.
On every pull request	A GitHub Action (GitLab equivalent included) measures each PR automatically and posts the summary as a comment on the PR itself — where reviewers already look. Can be set to block a merge on your policy.
Command line & CI	One command for engineers and pipelines (Jenkins, Docker, scheduled runs) — same numbers, headless.
What comes out	Human-readable HTML reports and a diff viewer, plus CSV/JSON for spreadsheets and dashboards, and SARIF for the GitHub Security tab. Everything runs inside your own infrastructure; no code leaves your machines.

The AI layer

AI agent	Software that calls an AI model and acts on the answer — running commands, writing files, sending requests. In your code or hidden in a dependency.
Agent Scan	Static detection of the AI layer: named agent SDKs and model calls, model output reaching execution (the dangerous pattern), and agent infrastructure left in the tree — workspaces, configs, committed credentials. Evidence with file and line, never guesswork.
AI-BOM	The AI Bill of Materials: an inventory of every place your software touches AI — which models, whose servers, what the output can reach, where data goes. Native or CycloneDX format; the artifact auditors and the EU AI Act ask about.
AI audit	The optional ML-based estimate of how AI-authored code *looks*. Off by default, and always labelled what it is: pointers for review, never verdicts — per-file AI authorship detection is not reliably possible, and we publish the evidence for that.
Agent trailers	When some AI tools commit code they append a machine-readable co-author line to the commit message. CodeDelta finds them by exact-pattern scan of the git history — deterministic, no guesswork. The count is a *floor* on AI involvement, never a total: the lines are voluntary and removable.

> *Every figure quoted here traces to a published, reproducible measurement — sources for all of it on the evidence page.*

SEE THESE NUMBERS FOR YOUR OWN CODEBASE

CodeDelta runs on macOS, Linux and Windows, free to try — or on every pull request.
Download CodeDelta → · Questions or licensing: **contact codedelta.app**