code_delta
SEE THE RISK. PREVENT THE DISASTER.

CODEDELTA · MANAGEMENT BRIEF

Financial justification

How CodeDelta can save you time and money!

Engineering is your company's biggest discretionary spend, and it's the only major expenditure where the budget holder cannot see precisely what happened. A director can see what marketing spent per lead and what sales closed. Ask what engineering did last quarter and the answer comes back as story points, tickets, or a feeling.

This isn't because engineers hide anything. The instruments are pointed at the wrong thing: plans, promises and activity, never the asset itself. The record of what happened to the code has existed all along — take two versions of a system and classify every changed statement as added, deleted, or edited in place. That classification is called **churn**, and it's the missing ledger: where the effort actually went, what got built versus rebuilt, and what keeps being redone — every entry checkable against the source.

That's "how does it help our business" in one breath: **it makes the biggest invisible budget visible.** In the AI era the case gets stronger, not weaker — machines multiplied the volume of change precisely when nobody could keep track of it by eye.

Three lines a director can read

The waste line — rework. "You paid for this twice." Some of every quarter's work replaces work already paid for once. A measured share of that is health — repair is what maintenance looks like — but the dial has bands, and a reading far off the bands is news (see overleaf).

The risk line — recurrence. "This keeps being redone — why?" A module that churns period after period is either where the business is, or a design fault that has been quietly billing you for years. Either way it was invisible until it was measured.

The governance line — the watchlist. "Your critical systems were touched N times." Every business has code it regards as settled: billing, the ledger, the module that passed the audit. A change ledger is the only instrument that reports those systems were touched at all, because the people touching them rarely think of it as news.

The questions these produce are the point: *"Why is this one piece of the system being changed every day?"* and *"That module is mission-critical — why was it worked on at all?"* They're budget-review questions, asked in plain English, with numbers behind them.

What the numbers look like

Rework is the share of changed statements that were edited in place rather than deleted and replaced — the repair rate of your process. Measured across 136 codebases and 393 million statements: hand-maintained projects at mid size repair about one changed statement in six; the largest codebases, one in ten to one in twenty. A heavily agent-built codebase measured one in five hundred — code that grows fast and is repaired by nobody. A decade of Kubernetes shows healthy maturing software moving the other way: repair climbing from 0.5% in its growth spurt to 11.6% at maturity.

Method, bands and per-measurement evidence are published at codedelta.app/papers.html; every figure above is cited there, with commit hashes, so your team can reproduce it.

A worked example

Suppose a forty-engineer organisation, fully loaded at $200,000 a head: an eight-million-dollar line item. Suppose the ledger shows a quarter of the year's repair work landing in one payments module. Nobody needed a metric to spend that money — but nobody could see it being spent, either. The concentration, once visible, is a question that asks itself in the next budget review. (This scenario is arithmetic, offered as an illustration; the measured figures in this brief are the cited ones above.)

The instrument

CodeDelta reads two versions of your code — a single pull request, or 43.5 million lines of Chromium on one laptop — and writes the ledger at statement level. Engineers get the detail: a Code Browser with every file, class and function ranked by churn and by repair. The pipeline gets a GitHub Action that writes a ledger entry on every pull request and can block a merge on policy. Management gets the trend across runs: direction, quarter on quarter, on the code itself. It also maps AI exposure — which files call out to model providers, and where the data would go — reported as pointers for review, not verdicts.

Deploying CodeDelta to make a difference

Start where your code already lives. On GitHub, add the CodeDelta Action from the Marketplace and every pull request writes its own ledger entry from the next PR onwards. Everywhere else, the engine ships as a Docker image and a plain binary bundle, so it drops into GitLab CI, Jenkins, Azure DevOps, Kubernetes or any self-hosted runner the same afternoon — and it all runs inside your own infrastructure. There's no upload and no SaaS: your code never leaves your network, which is the first question a bank's security team asks and the answer they want. There's a desktop route too — macOS, Linux and Windows — point it at the two most recent releases of your own system and the first report is the ledger for your last release cycle. Free to evaluate on every path. Download and documentation: **codedelta.app**.