WHITEPAPER · DELIVERY

From Spec to Shipped

Measuring delivery against requirements, so "done" means the plan shipped, not just that the tickets closed.

CodeReviewer
A product of Techware Lab · codereviewer.cloud · September 2026

CONTENTS

What’s inside

SECTION 01

"Done" is a claim, not a measurement

Ask a team whether a feature is finished and you will usually get an answer based on activity: the tickets are closed, the pull requests are merged, the branch is deployed. Those are signals that work happened. They are not proof that **the thing that was promised** now exists.

The gap between the two shows up at the worst possible moment: in the client demo, in user acceptance testing, or in a support ticket weeks after launch. A requirement that was agreed, estimated and scheduled simply never made it into the code, and nobody noticed because every individual task looked complete.

This paper argues that delivery should be measured against the specification itself, continuously, rather than inferred from task status at the end. It describes where requirements get lost, a lightweight way to trace them, the metrics that matter, and a model for the cost of discovering gaps late.

KEY IDEA

The core claim

Progress should be measured requirement by requirement, with evidence from the code, not ticket by ticket, with evidence from a status field.

Who this is for

Agencies and delivery teams building to a client specification feel this most sharply, because the spec is effectively the contract. But any product team that writes requirements before building benefits from knowing, at any moment, which ones are shipped, which are half-done and which have been forgotten.

SECTION 02

Where requirements go missing

Requirements rarely disappear through a single mistake. They leak out at each hand-off in a chain that runs from the specification to the merged code.

Hand-off	How requirements leak
Spec to tasks	A requirement is folded into a larger task and its details are summarised away.
Tasks to pull requests	A task is closed when most of it is done; the remainder is never ticketed.
Within a pull request	An edge case in the spec (refunds, other currencies, empty states) is skipped.
Across pull requests	Two people each assume the other handled a shared requirement.
Scope changes	A requirement is quietly dropped or changed without the spec being updated.

Partial delivery is the dangerous case

Requirements that are entirely missing are relatively easy to spot in a demo. Requirements that are *partially* delivered are much harder: the screen exists, the button works, but the export omits refunds or the reminder email never fires for yearly plans. These pass a casual walkthrough and fail in production.

Why reviews do not catch it

Code review looks at a single change in isolation. A reviewer checking that a pull request is correct is not usually checking that it is *complete* relative to a document they may never have read. Completeness is a property of the whole body of work, and no individual review is positioned to see it.

SECTION 03

Lightweight requirements traceability

Traditional traceability matrices are heavy, manual and rarely kept up to date. A lighter approach keeps the useful core: every requirement has an identity, a priority and a status backed by evidence.

1. Write requirements as discrete, checkable statements

"Invoices download as CSV" is checkable. "Improve billing" is not. Each requirement should describe an observable outcome a reviewer could confirm in the code or the product.

2. Prioritise with must, should and may

Not all requirements carry equal weight. Separating must-haves from should-haves and nice-to-haves lets the team report honestly on what matters: "all must-haves shipped, two shoulds outstanding" is a far more useful status than a single percentage.

3. Attach evidence to every status

A requirement is marked shipped only when it can point at the change that delivered it. Evidence turns a status from an opinion into something anyone can verify.

Status	Meaning	Evidence
Shipped	The requirement is fully delivered.	Merged change(s) that implement it.
Partial	Some of the requirement is delivered.	The change, plus what is still missing.
Missing	No change delivers it yet.	None. This is the gap list.

SECTION 04

Metrics that tell you whether you are done

With requirements traced, a handful of metrics answer the question stakeholders actually ask: "are we done?"

Metric	Definition	Use
Requirement coverage	Shipped requirements / total requirements.	Headline progress.
Must-have coverage	Shipped musts / total musts.	Release readiness.
Partial count	Requirements with status partial.	Hidden risk; review these first.
Gap age	Time a requirement has been missing since work began.	Finds the forgotten ones.
Evidence ratio	Shipped requirements with linked evidence.	Trust in the numbers.

Report by priority, not a single number

A single percentage can hide the one must-have that blocks launch behind many completed nice-to-haves. Always show must-have coverage separately, and list partial and missing musts by name.

EXAMPLE

A useful weekly status line

"Billing revamp: 7 of 9 requirements shipped; all musts complete; bulk export is half-done; yearly proration not started." Every clause is backed by evidence.

SECTION 05

The cost of finding gaps late

The cost of a missed requirement depends heavily on when it is found. Found while the code is fresh, it is a small follow-up. Found at the demo, it means reopening finished work, re-testing and often renegotiating dates.

ILLUSTRATIVE: REPLACE WITH YOUR OWN NUMBERS

Early versus late discovery of missed requirements

Assumption	Value
Requirements in the specification	40
Share missed or partially delivered	10%
Hours to fix when found during development	1.0
Hours to fix when found at demo or acceptance	8.0
Loaded cost per engineering hour (USD)	$60

MODEL

Formula

Missed requirements = requirements x miss share. Cost = missed requirements x fix hours x hourly cost, evaluated once with early fix hours and once with late fix hours.

Scenario	Hours	Cost
Found during development	4	$240
Found at demo or acceptance	32	$1,920
Difference per project	**28**	**$1,680**

The engineering hours are only part of the picture. Late discovery also costs schedule, client confidence and, for agencies, margin on fixed-price work. The model leaves these out deliberately; if you include them, the case for early detection only gets stronger.

Multiply the difference by the number of projects or features you deliver in a year to see the annual effect for your organisation.

SECTION 06

A delivery playbook

- **Keep the spec in one place, in plain text.** A single Markdown document per feature is easy to version, diff and review.
- **Give every requirement a clear, checkable statement and a priority.**
- **Link each pull request to the requirements it delivers.** Make it part of the PR description.
- **Review the gap list weekly.** Partial and missing musts are the agenda.
- **Update the spec when scope changes.** A dropped requirement should be a decision, not an accident.
- **Show the scorecard to the client.** Shared evidence replaces status meetings.

Checklist

- [ ] Every feature has a written specification before development starts.
- [ ] Requirements are discrete, checkable and marked must, should or may.
- [ ] Each requirement status is backed by a linked change.
- [ ] Partial requirements are tracked separately from shipped ones.
- [ ] Must-have coverage is reported on its own, by name.
- [ ] Scope changes are reflected in the spec the same week.

SECTION 07

How CodeReviewer helps

CodeReviewer measures delivery against the spec automatically, as the work happens.

- **Upload the spec.** A Markdown specification is split into individual must, should and may requirements.
- **Every pull request is checked against it.** Requirements are matched to the changes that deliver them, with evidence you can click through.
- **A live scorecard.** See what is shipped, what is half-done and what has been forgotten before the demo, not during it.
- **Spec-aware merging.** Optionally, only merge automatically when a change matches the spec.

NEXT STEP

See your own scorecard

Start a 15-day free trial at codereviewer.cloud, upload a spec, and watch it fill in as pull requests land.

Give every PR the review it deserves.

Start a 15-day free trial. No card needed.

codereviewer.cloud