Stonetusker Insights · Self-assessment: software delivery

Where Does Your Delivery Wait? Five Numbers to Check in 20 Minutes

A self-assessment that uses systems you already have. Five numbers, a simple way to score them and what each result usually means.

For: Any team with Git and a build pipeline.

By Subeesh Sivanandan, Founder, Stonetusker Systems · Updated 29 October 2026 · About 4 minutes to read · PDF version

The short version

- Delivery speed depends mostly on the time work spends waiting between steps, not on how fast code is written.
- Five numbers show where the waiting happens: pull request waiting time, merge to production, pipeline count, rerun rate and manual release steps.
- One person with read access can produce them for one repository in about 20 minutes, with nothing to install.
- Score each number, pick the weakest one, change one thing and measure again next quarter.

5	20 minutes	50%
Numbers that show where delivery work waits	To produce them for one repository, with nothing to install	Developers who lose 10 or more hours a week to friction outside coding[4]

Why it matters now

Most organisations judge delivery by activity, such as commits, tickets closed and the output of coding assistants. A change usually spends hours being written and days waiting for a reviewer, a pipeline, a release window or an approval. Faster writing reaches customers only when those waits shrink too.

Developers feel the waiting. Atlassian's 2025 survey of 3,500 developers and managers found that half of developers lose 10 or more hours a week to friction outside coding.[4] The five numbers below measure that waiting and show which step causes most of it.

Run the checkup in 20 minutes

Pick one repository that matters and the last full quarter. You need read access on GitHub or GitLab, the GitHub command-line tool (`gh`) or `curl`, and `jq`. Your data stays on your own machine.

Step 1. Pull request waiting time

Measure the median time from opened to merged and the median time to first review.[1]

```
# Median hours from pull request opened to merged
gh pr list --repo ORG/REPO --state merged --limit 1000 \
  --search "merged:2026-07-01..2026-09-30" --json createdAt,mergedAt
\
  | jq '[.[] | select(.mergedAt != null and .createdAt != null)
        | ((.mergedAt | fromdateiso8601) - (.createdAt |
fromdateiso8601)) / 3600]
        | sort | .[length / 2 | floor]'

# Median hours to first review
gh pr list --repo ORG/REPO --state merged --limit 1000 \
  --search "merged:2026-07-01..2026-09-30" --json createdAt,reviews
\
  | jq '[.[] | select(any(.reviews[]; .submittedAt != null))
        | ((.reviews | map(select(.submittedAt != null) |
.submittedAt | fromdateiso8601) | min)
           - (.createdAt | fromdateiso8601)) / 3600]
        | sort | .[length / 2 | floor]'
```

On GitLab, `/api/v4/projects/:id/merge_requests?state=merged` returns `created_at` and `merged_at` for the same calculation.[3] If time to first review is more than half of the total, pull requests mostly wait for a reviewer.

Step 2. Time from merge to production

Few teams measure this, and it is often the largest wait. If you tag releases, run this inside a clone to see when the last 20 merged pull requests reached a tag.

```
gh pr list --repo ORG/REPO --state merged --limit 20 --json
number,mergedAt,mergeCommit \
  | jq -r '.[] | "\(.number) \(.mergedAt) \(.mergeCommit.oid)"' \
  | while read n merged sha; do
      tag=$(git tag --contains "$sha" --sort=creatordate | head -1)
      when=${tag:+$(git for-each-ref --format='%
(creatordate:iso8601)' "refs/tags/$tag")}
      echo "PR $n merged $merged released ${tag:-not yet} $when"
    done
```

Without tags, use the deployment history of your deployment tool. A tag may differ from a production release, so compare the result with how you actually release.

Step 3. The number of ways you build and ship

Place your active repositories in one folder and count pipeline definitions.

```
find . \( -path '*/.github/workflows/*.yml' -o -path
'*/.github/workflows/*.yaml' \
  -o -name '.gitlab-ci.yml' -o -name 'Jenkinsfile' -o -name 'azure-
pipelines.yml' \
  -o -name 'bitbucket-pipelines.yml' -o -path
'*/.circleci/config.yml' \) | wc -l
```

Then record the share of repositories built from a shared template: reusable workflows (`uses: ORG/some-repo/.github/workflows/...`) on GitHub, or `include:` entries that point to a shared project on GitLab.

Step 4. Pipeline rerun rate

Reruns are the clearest sign of unstable tests or infrastructure. On GitHub Actions:[2]

```
gh run list --repo ORG/REPO --limit 500 --json attempt \
  | jq '{runs: length, reruns: ([.[] | select(.attempt > 1)] |
length)}
        | .rate = (.reruns / .runs * 100 | floor)'
```

Step 5. Manual release steps

List every step a person performed between merge and production in the last release: approvals, tickets, change meetings, manual tests and scripts run from a laptop. Count them, and mark which ones actually catch problems.

Score your five numbers

Measure	Needs attention	Workable	Strong	A weak score usually points to
Median time from pull request opened to merged	More than 3 days	1 to 3 days	Less than 1 day	Review capacity and team habits
Median time from merge to production	More than 1 week	1 day to 1 week	Same day	Release batching
Repositories on shared pipeline templates	Below 30%	30% to 70%	Above 70%	Pipeline standardisation
Pipeline rerun rate	Above 10%	3% to 10%	Below 3%	Test and infrastructure stability
Manual release steps	More than 8	3 to 8	0 to 2	Release automation

These bands are working rules of thumb for growth-stage companies, meant to find the weakest area. They are not an industry benchmark.

To express the waiting as a cost, multiply the pull requests merged in a quarter by the median hours to first review, and count a modest share (for example 10%) as lost time. With 1,200 pull requests, a 20-hour median and 10%, that is 2,400 engineering hours a quarter, close to 10,000 hours a year.

Act on the result, step by step

1. **Choose the scope.** Pick one important repository and the last full quarter, and note the same quarter last year for comparison.
2. **Produce the five numbers.** Run the five checks above and write down each result.
3. **Score them.** Place each number in the scoring table and find the weakest area.
4. **Change one thing.** Choose one change that addresses the weakest area, with an owner and a date.
5. **Measure again.** Repeat the checkup next quarter. The trend matters more than any single result.

Manual approvals often exist for SOC 2 or ISO 27001 change controls. Auditors look for changes that are reviewed, tested and traceable, which pull request reviews, required checks and deployment logs usually show. Agree any change with your auditor first.

What is still open

- The scoring bands are rules of thumb. Your industry, release model and regulatory context may justify different targets.
- The cost figure depends on the share of waiting you count as lost time. The 10% is an illustrative assumption.
- Git and pipeline history show waiting well. Customer impact and incident cost need other data.

A note to forward to your team

Delivery checkup

I would like to know where our delivery waits. Please run the five checks in this note on [repository] for [quarter] by [date] and reply with five numbers:
1. Median hours from pull request opened to merged (and to first review)
2. Median time from merge to production
3. Share of repositories on shared pipeline templates
4. Pipeline rerun rate
5. Manual steps in the last release

Owner: [name]
Note: https://stonetusker.com/insights/delivery-checkup.html

Questions

Do we need a paid tool to measure delivery?

Git hosting and pipeline systems already hold the data. The GitHub command-line tool or the GitLab interface, together with jq, is enough for a first measurement.

Why use the median instead of the average?

A few very old pull requests or delayed releases distort an average. The median shows what a typical change experiences.

How do these numbers relate to the DevOps Research and Assessment measures?

They overlap. Time from merge to production is part of lead time for changes, and the other numbers explain why lead time is long. Change failure rate is a useful sixth measure if you track incidents.

How often should we run the checkup?

Once a quarter suits most teams. Improving one constraint at a time and measuring again works better than small changes everywhere.

Want a second opinion?

Run the checkup yourself first. If you would like a second opinion, reply with your five numbers only. No repository names, code or command output are needed. Within five working days you receive a short written review showing where your delivery waits, the likely causes and the first change Stonetusker would make. It is free and confidential, and a call is optional.

Ask for a second opinion

The button opens an email to subeesh@stonetusker.com with the subject and the numbers to fill in. You can also use the contact page, or reply to the email or LinkedIn message that brought you here.

Next page for your situation

- Coding assistants and your delivery baseline: compare the same numbers before and after coding assistants.
- Peak-season release readiness: if your business has a seasonal peak.

Sources

1. GitHub CLI manual: gh pr list
2. GitHub CLI manual: gh run list
3. GitLab: Merge requests API
4. Atlassian: State of Developer Experience Report 2025

About the author

Subeesh Sivanandan is the Founder and CEO of Stonetusker Systems. He has worked for more than 27 years in release engineering, DevOps, platform engineering & embedded Linux. His career includes serving as the Principal Engineer & Manager of Release Engineering at Stryker, alongside earlier roles at Nokia, VeriSign, and IP Infusion. LinkedIn profile.

Revision history (recent 3 updates)

Version 2.1, 29 October 2026. Every fact on this page is checked against a minimum of two sources each month.

Version	Date	Change
1.6	16 October 2026	Rewritten as an article for quick readers: introduction, summary of all major changes, what it means and first steps; one figure; commands folded for platform leads; print option and direct email offer
2.0	25 October 2026	Simpler layout and shorter text; the review offer now asks for a few counts only
2.1	29 October 2026	Who-this-is-for box, email offer with the numbers listed, and next pages