Trunk-Based Development, Feature Flags and DORA

Version DORA 2025 · verified 2026-11-02

Merge to trunk daily and keep it releasable — short branches, feature flags, branch by abstraction, CI rules, DORA metrics and the AI Capabilities Model.

Contents

Mental model 3

Trunk vs long-lived branches 3

Short-lived branches and stacking 4

Feature flags 6

Branch by abstraction 7

CI requirements 9

Release vs deploy 11

DORA's delivery metrics 12

The 2025 AI Capabilities Model 13
Failure modes 15
Further reading 16

Mental model

Integration frequency is the lever. The longer work stays off trunk, the bigger and riskier the merge, the slower the review, and the later you learn it does not fit — so integrate small batches at least daily. Feature flags, fast CI, branch by abstraction and small reviews are not separate practices; each exists to make merging unfinished work every day safe. DORA's metrics then tell you whether it is working.

Trunk vs long-lived branches

DORA practice	Target
Active branches in the repo	Three or fewer
Merge to trunk	At least once a day
Code freezes, integration phases	None

In trunk-based development each developer splits work into small batches and merges into trunk (also called `main` or mainline) at least once a day. Feature-branch workflows keep several people on a branch for days or

weeks, then pay for it with large merges, conflict resolution and stabilisation periods. DORA's analysis of its 2016 and 2017 data links the three targets above to higher delivery speed, stability and availability.

```
Long-lived: main --------------*  big merge
            feat  *--*--*-----/   (2 weeks)
Trunk:      main --*--*--*--*--*  daily merges
```

> **Gotcha:** "We use pull requests, so we can't do trunk-based" is wrong. A PR branch that lives hours, not weeks, *is* trunk-based. Age of the branch is the measure, not whether a branch exists.

Short-lived branches and stacking

Rule	Why
One developer (or a pair) per branch	No team branch to drift

Rule	Why
Lives hours, at most about two days	Merge stays trivial
Build passes before merge	Trunk stays releasable
Stack dependent PRs	Next step need not wait

trunkbaseddevelopment.com allows short-lived feature branches for review, kept to a couple of days and one developer or pair; DORA describes branches that typically last no more than a few hours. When step two depends on unmerged step one, branch from step one and open a stacked PR so neither blocks.

```
main <- PR #1 schema + tests
         <- PR #2 service uses schema
            <- PR #3 UI behind flag
merge order: #1, then #2, then #3
```

Feature flags

Type	Lifetime	Example
Release	Days to weeks	Hide unfinished checkout
Experiment	Length of the test	A/B a pricing page
Ops	Can be long; runtime switch	Kill switch for a slow call
Permissioning	Months to years	Premium-only feature

The Hodgson/Fowler taxonomy sorts flags by how long they live and how dynamic the decision is. Release flags let unfinished code merge and deploy dark; they are inventory and must be deleted once the feature is on for everyone. Ops flags must be switchable without a release — that is their whole point during an incident.

```
type Flags = { newCheckout: boolean }
const flags: Flags = { newCheckout: false }
```

```
// RELEASE flag — owner: @payments
// remove by 2026-12-01 (ticket PAY-88)
const checkout = (cart: string[]) =>
  flags.newCheckout
    ? `v2:${cart.length}`
    : `v1:${cart.length}`

checkout(['mug'])  // "v1:1"
```

> **Warning:** Every release flag doubles the paths to test and lingers unless someone owns its removal. Give each flag an owner and an expiry when you create it, and fail the build on expired flags.

Branch by abstraction

Step	Trunk state
1. Introduce an abstraction over the old code	Green, same behaviour

Step	Trunk state
2. Move callers to the abstraction	Green
3. Build the new implementation behind it, off	Green, not used
4. Switch on (flag or config)	Green, new path live
5. Delete old implementation and, if useless, the abstraction	Green

Large changes — replacing an ORM, a payment provider, a rendering engine — are where long-lived branches usually come back. Branch by abstraction does the replacement **on trunk** in small commits, each keeping the build green and releasable, so the change can pause or roll back at any step.

```
interface Search {
  find(q: string): string[]
```

```
}
const legacy: Search = {
  find: (q) => [`sql:${q}`] }
const engine: Search = {
  find: (q) => [`idx:${q}`] }

const useEngine = false  // step 4 flips this
const search = useEngine ? engine : legacy
search.find('mug')  // ["sql:mug"]
```

CI requirements

Requirement	Detail
Tests run before every merge	Locally and in CI
Build and tests take minutes	DORA: feedback in under ten minutes
Red trunk is everyone’s problem	Fix in minutes, or revert
Developers own the tests	Not a separate QA team

Requirement	Detail
Review is fast	Heavy review breaks small batches

DORA defines continuous integration as trunk-based development plus a fast automated test suite run on every commit to trunk. If CI fails, developers stop and fix it immediately, or revert if the fix takes more than a few minutes. A slow build pushes people to batch changes, which undoes the practice — treat build time as a defect.

```
trunk red at 10:42 (commit a1b9)
10:44 author: not a 5-minute fix
10:45 revert a1b9 -> trunk green
      re-land later with a test
```

Release vs deploy

Term	Means
Deploy	New code is running in production
Release	Users can see the new behaviour
Dark launch	Deployed, flag off
Progressive release	Flag on for 1%, 10%, 100%

Flags decouple the two events. Deploying becomes a routine, low-risk technical act done many times a day; releasing becomes a product decision you can make, stage and undo without a deploy. A rollback of a release is a flag flip, which is faster than any redeploy.

```
Mon  deploy   checkout v2   flag off
Tue  release  internal staff only
Wed  release  5% of users, watch errors
Thu  release  100%
+2w  delete flag and v1 code path
```

DORA's delivery metrics

Metric	Measures
Change lead time	Commit to running in production
Deployment frequency	Deploys per period
Failed deployment recovery time	Time to recover from a failed deploy
Change fail rate	Deploys needing immediate intervention
Deployment rework rate	Unplanned deploys caused by incidents

DORA moved from the original four keys to five metrics, grouped as **throughput** (the first three) and **instability** (the last two). Its research shows speed and stability are not a trade-off: top performers do well on all five. Measure per application or service, not blended across an organisation, and start from conversations or the DORA Quick Check before building dashboards.

```
Honest measurement
- per service, trend over time
- deploys = production deploys only
- "failed" = needed rollback or hotfix
- never a target, never a team ranking
```

> **Gotcha:** Making a metric a goal (“everyone deploys daily by Q4”) invites gaming — empty deploys, incidents not labelled as failures. DORA names this, single-metric thinking and team comparisons as the common pitfalls.

The 2025 AI Capabilities Model

Capability	Amplifies AI by
Clear and communicated AI stance	Removing ambiguity on tools and use
Healthy data ecosystems + AI-accessible internal data	Giving AI good, reachable context

Capability	Amplifies AI by
Strong version control practices	Frequent commits, routine rollback
Working in small batches	Keeping AI output reviewable
User-centric focus	Pointing speed at user value
Quality internal platforms	Scaling gains across teams

The 2025 DORA report's finding is that AI is an amplifier: it raises throughput but still correlates with lower stability, and teams with strong automated testing, mature version control and fast feedback loops benefit while others mostly see more instability. Its companion AI Capabilities Model names seven capabilities (the two data ones share a row above). Version control and small batches are this sheet's practices; the rest set the context they run in.

```
AI writes more code per day
  -> small batches keep each change reviewable
  -> trunk + CI catch breakage in minutes
  -> flags + rollback limit blast radius
```

Failure modes

Symptom	Real cause
“Trunk-based” with week-old PRs	Branch age not tracked
Hundreds of stale flags	No owner or expiry
Trunk red for hours	No fix-or-revert rule
Big-bang merges for “big” work	Branch by abstraction not used
Release train with a freeze	Release and deploy still coupled

Most failures are a practice adopted in name only. Measure branch age, open flag count and time-to-green alongside the DORA metrics; they are the leading indicators the five metrics lag behind.

```
Weekly check
oldest open PR        < 2 days
release flags > 30d   0
trunk red time        < 30 min total
```

Further reading

- DORA — Trunk-based development
- DORA — Software delivery performance metrics
- DORA — AI Capabilities Model
- trunkbaseddevelopment.com
- Martin Fowler — Feature Toggles