RefactorOps

Definition of Ready / Definition of Done Pack

Practical systems to create clarity before the work starts and accountability before the work ends.

By Michael Smith — Founder, RefactorOps
"Turning engineering chaos into profit."

Why this matters

Most delivery chaos is not caused by bad developers. It's caused by unclear work. Vague stories get pulled into a sprint, then everyone argues about what "done" means at the end. People burn nights and weekends trying to close work that should never have started in the first place.

Definition of Ready (DoR) is the line for "this work is actually ready to start." **Definition of Done (DoD)** is the line for "this work is actually complete."

Used correctly, these two guardrails do three things:

- They prevent half-baked work from entering a sprint.
- They kill most of the "almost done" lies at sprint review.
- They make quality predictable without adding meetings.

When teams say "agile doesn't work," what they usually mean is: "We never agreed on what *ready* or *done* means."

Who owns DoR and DoD?

Your engineering org should publish a **baseline Definition of Ready** and a **baseline Definition of Done**. That baseline is the standard. Everyone should be familiar with it.

Each team is then allowed, expected really, to **adapt** those checklists to match their reality. A front-end team will have accessibility and browser coverage in “done.” A data integration team won’t.

Rule of thumb: **You can cut noise. You can add detail. You cannot ignore the concept.**

> **Good behavior:** “We copied the org DoD, removed browser matrix because we’re API-only, and added encryption checks because we handle PHI.”
>
> **Bad behavior:** “We don’t really use DoR here; we just pull tickets and talk it out.” That’s how you end up working weekends.

How to use this in real life

1. DURING BACKLOG REFINING

You apply the **Definition of Ready**. If a ticket meets DoR, tag it as **Ready** (or whatever your tool calls that state). If it doesn’t, it stays in backlog. No debate. No exceptions.

Don’t spin up a sign-off meeting. Just tag it. Ready / Not Ready is all you need.

2. DURING SPRINT PLANNING

Work that wasn’t “Ready” doesn’t get pulled in unless you’re doing intentional discovery/spike work. You’re protecting your team from chaos someone else didn’t bother to resolve.

3. DURING SPRINT REVIEW / ACCEPTANCE

You apply the **Definition of Done**. If a story doesn’t meet DoD, it is not “Done.” It does not get demoed as complete. It rolls forward.

This is where predictable delivery comes from. Not burndown charts. Standards.

> Don’t weaponize this. DoR and DoD are not political leverage. They’re clarity tools. If someone misses something, fix the process and move on.

About design / Figma / content references

Designs change. Marketing changes their mind. Stakeholders “tweak” things after approval.

When you tag a story as Ready, attach a screenshot or export of the design/UX/content reference **as it existed at that moment**. Don’t just paste a Figma link.

Links move. Screenshots don’t. Screenshots save you when someone says “that’s not what we agreed to.”

Definition of Ready (DoR) Use this in backlog planning

A ticket or story is “Ready” when the team can start execution without guessing, waiting on someone else, or filling in business logic on the fly.

✔	ITEM	OWNER / SOURCE	NOTES
☐	**Business goal is clear**	Product Owner / BA	We know why we’re doing this, not just what it is.
☐	**Acceptance criteria are written and testable**	QA + Product Owner	No “TBD,” no “as discussed.” AC is specific and binary.
☐	**Dependencies are identified**	Tech Lead	APIs, credentials, content, approvals, legal, etc.
☐	**Design / UX reference attached (screenshot or export)**	Designer / Content	Don’t just link to Figma. Paste the version you’re building.
☐	**Technical approach agreed**	Developer + Tech Lead	No mystery tech. Dev knows roughly how they’ll build it.
☐	**No blockers remain**	Team	We’re not waiting on access, data, approvals, or direction.
☐	**Story is estimable**	Team	Everyone agrees this fits inside a sprint/iteration.

If a ticket doesn’t meet DoR, don’t pull it. Don’t “be a hero.” Heroes cause rework.

Definition of Done (DoD) Use this in sprint review / acceptance

Work is “Done” when it is shippable and nobody has to circle back later to quietly finish it. “It’s basically done” does not count.

✔	ITEM	OWNER / ACCOUNTABLE	NOTES
☐	**Code merged and reviewed**	Developer / Peer reviewer	No local branches, no “I’ll PR it later.”
☐	**Acceptance criteria verified**	QA	AC isn’t “generally works.” AC is pass/fail. QA confirms pass.
☐	**Accessibility & performance meet baseline**	Front-end / Tech Lead	Whatever your baseline is — document it. Enforce it.
☐	**Test coverage in place**	Developer	Unit/integration where it matters. No critical path untested.
☐	**Deployed to the correct environment for review**	Developer / Tech Lead	Staging or equivalent, not “it works on my machine.”
☐	**Demo/Reproduction steps clearly outlines**	Developer / Tech Lead	Exact steps for happy path demo documented on ticket.
☐	**No critical known issues remain**	Team	We aren’t sweeping broken edge cases under the rug to “hit velocity.”
☐	**Docs / changelog updated**	Developer	Someone in 3 months should understand what changed and why.
☐	**Product owner / stakeholder validation**	Product Owner / Client Lead	They’ve seen it and said “Yes, that solves what I asked for.”

If it doesn’t pass DoD, it’s not done. Call it what it is. Roll it forward.

What to do next

Step 1: Take these checklists and make a copy for your team.
Step 2: Cut the noise. Add what's missing.
Step 3: Publish it somewhere everyone can see.
Step 4: Use DoR at grooming. Use DoD at review. Every sprint.

If you want this installed across your team in a day, including backlog triage, QA standards, and stakeholder alignment, that's what the **RefactorOps Foundations of Technical Leadership Workshop** is for.

RefactorOps helps technical organizations eliminate waste, fix broken delivery, and operate like profit centers, not cost centers.

RefactorOps

Turning engineering chaos into profit.