The Dangers of a Definition of Ready

by Mike Cohn • 64 Comments

Although not as popular as a Definition of Done, some Scrum teams use a Definition of Ready to control what product backlog items can enter an iteration.

You can think of a Definition of Ready as a big, burly bouncer standing at the door of the iteration. Just as a bouncer at a nightclub only lets certain people in—the young, the hip, the stylishly dressed—our Definition-of-Ready bouncer only allows certain user stories to enter the iteration.

And, as each nightclub is free to define who the bouncers should let into the club, each team or organization is free to define its own definition of ready. There is no universal definition of ready that is suggested for all teams.

A Sample Definition of Ready

So what types of stories might our bouncer allow into an iteration? Our bouncer might let stories in that meet rules such as these:

- The conditions of satisfaction have been fully identified for the story.
- The story has been estimated *and* is under a certain size. For example, if the team is using story points, a team might pick a number of points and only allow stories of that size or smaller into the iteration. Often this maximum size is around half of the team's velocity.
- The team's user interface designer has mocked up, or even fully designed, any screens affected by the story.
- All external dependencies have been resolved, whether the dependency was on another team or on an outside vendor.

A Definition of Ready Defines Pre-Conditions

A Definition of Ready enables a team to specify certain pre-conditions that must be fulfilled before a story is allowed into an iteration. The goal is to prevent problems before they have a chance to start.

For example, by saying that only stories below a certain number of story points can come into an iteration, the team avoids the problem of having brought in a story that is too big to be completed in an iteration.

Similarly, not allowing a story into the iteration that has external dependencies can prevent those dependencies from derailing a story or an entire iteration if the other team fails to deliver as promised.

For example, suppose your team is occasionally dependent on some other team to provide part of the work. Your user stories can only be finished if that other team also finishes their work—and does so early enough in the iteration for your team to integrate the two pieces.

If that team has consistently burned you by not finishing what they said they'd do by the time they said they'd do it, your team might quite reasonably decide to not bring in any story that has a still-open dependency on that particular team.

A Definition of Ready that requires external dependencies to be resolved before a story could be brought into an iteration might be wise for such a team.

A Definition of Ready Is Not Always a Good Idea

So some of the rules our bouncer establishes seem like good ideas. For example, I have no objection against a team deciding not to bring into an iteration stories that are over a certain size.

But some other rules I commonly see on a Definition of Ready can cause trouble—big trouble—for a team. I'll explain.

A Definition of Ready can be thought of like a gate into the iteration. A set of rules is established and our bouncer ensures that only stories that meet those rules are allowed in.

If these rules include saying that something must be 100 percent finished before a story can be brought into an iteration, the Definition of Ready becomes a huge step towards a sequential, stage-gate approach. This will prevent the team from being agile.

A Definition of Ready Can Lead to Stages and Gates

Let me explain. A stage-gate approach is characterized by a set of defined *stages* for development. A stage-gate approach also defines *gates*, or checkpoints. Work can only progress from one stage to the next by passing through the gate.

When I was a young kid, my mom employed a stage-gate approach for dinner. I only got dessert if I ate all my dinner. I was not allowed to eat dinner and dessert concurrently.

As a product development example, imagine a process with separate design and coding stages. To move from design to coding, work must pass through a design-review gate. That gate is put in place to ensure the completeness and thoroughness of the work done in the preceding stage.

When a Definition of Ready includes a rule that something must be *done* before the next thing can *start*, it moves the team dangerously close to stage-gate process. And that will hamper the team's ability to be agile. A stage-gate approach is, after all, another way of describing a waterfall process.

Agile Teams Should Practice Concurrent Engineering

When one thing cannot start until another thing is done, the team is no longer overlapping their work. Overlapping work is one of the most obvious indicators that a team is agile. An agile team should always be doing a little analysis, a little design, a little coding, and a little testing. Putting gates in the development process prevents that from happening.

Agile teams should practice *concurrent engineering*, in which the various activities to deliver

working software overlap. Activities like analysis, design, coding, and testing will never overlap 100%—and that's not even the goal. The goal is overlap activities as much as possible.

A stage-gate approach prevents that by requiring certain activities to be 100% complete before other activities can start. And a definition of ready can lead directly to a stage-gate approach if such mandates are included in the Definition of Ready.

That's why, for most teams, I do not recommend using a Definition of Ready. It's often unnecessary process overhead. And worse, it can be a large and perilous step backwards toward a waterfall approach.

In some cases, though, I do acknowledge that a Definition of Ready can solve problems and may be worth using.

Using a Definition of Ready Correctly

To use a Definition of Ready successfully, you should avoid including rules that require something be 100 percent done before a story is allowed into the iteration—with the possible exception of dependencies on certain teams or vendors. Further, favor guidelines rather than *rules* on your Definition of Ready.

So, let me give you an example of a Definition of Ready rule I'd recommend that a team rewrite: "Each story must be accompanied by a detailed mock up of all new screens."

A rule like this is a gate. It prevents work from overlapping. A team with this rule cannot practice concurrent engineering. No work can occur beyond the gate until a *detailed design* is completed for *each* story.

A better variation of this would be something more like: "If the story involves significant new screens, rough mock ups of the new screens have been started and are just far enough along that the team can resolve remaining open issues during the iteration."

Two things occur with a change like that.

1. The rule has become a guideline.

2. We’re allowing work to overlap by saying the screen mockups are are sufficiently far along rather than *done*.

These two changes introduce some subjectivity into the use of a definition of ready. We’re basically telling the bouncer that we still want young, hip and stylishly dressed people in the nightclub. But we’re giving the bouncer more leeway in deciding what exactly “stylishly dressed” means.

Posted: August 9, 2016

Tagged: product backlog, teams, backlog, definition of ready
