Get started Open in app

Steve Berczuk

Follow 172 Followers About

Prioritizing Agile Infrastructure

Steve Berczuk Jan 29, 2017 · 3 min read

Engineering practices such as Continuous Integration, Refactoring, and Unit Testing are key to the success of agile projects, and you need to develop and maintain some infrastructure to maintain these engineering practices. Some teams struggle with how to fit work that supports agile infrastructure into an agile planning process, and treat it differently that "features work." I advocate that infrastructure (and relatedly, technical debt), should be prioritized and managed in the backlog in the same way that feature work is.

Some teams attempt to draw a distinction between infrastructure and feature work by creating "infrastructure" or "technical" stories for tasks that are not directly tied to product features. In these stories the stakeholder may be a developer, and the product owner treats these backlog items as "overhead" or a "gift" to the team.

For example, a user story for setting up a continuous integration server can go something like:

> *As a developer I can ensure that all the code integrates without errors so that I can make steady progress.*

While not all work on a project leads directly to a user-visible feature, thinking of infrastructure stories differently than other stories has a number of risks, and can compromise the relationship between the development team and other stakeholders. Agile methods are about delivering value as measured by the Product Owner (though the team has input in helping the owner make the right cost/benefit decisions). Thinking about tools and as having value that is independent from end user value subjects you to the same risks that Big Design Up Front does: wasting effort and not delivering value.

I'm a fan of having everything an agile team does being related to creating value for the product owner. This may be a bit idealistic, but having this ideal as a starting point can be useful. Even if just try to focus on delivering value you will be more likely to make the right decisions for your team and your project.

One way to make the relationship between "infrastructure" work and product value clear is to recast any infrastructure stories with a focus on value to the product owner. For example, consider the example above for setting up a CI system:

> *As a project manager, I want to identify when changes break the code line so that mistakes can be fixed quickly, and the code stays healthy.*

This may not be the perfect example, but when I think of infrastructure items this way I have a better understanding of why I'm considering a change, and it prepares me to advocate for the work in case there is push-back. You can apply this approach to other "technical" items such as:

- Investigating a new testing framework.
- Refactoring (in the context of implementing a story).
- Upgrading your SCM system.
- Setting up a wiki
- Adding reporting to your Maven or Gradle build.

The benefits of considering the value of technical tasks to all stakeholders, and not just developers, include:

- A better relationship between engineers and the other stakeholders on a project.
- A better allocation of resources: If you can't explain the value of something there may be a better solution, or at least a less costly one.
- A better understanding of how to use engineering techniques to deliver value.

Johanna Rothman has written about applying a similar approach to technical debt, and in the end, missing infrastructure *is* technical debt.

I admit that his approach has some challenges. The value of technical tasks can be difficult to explain, and are often long-term at the expense of short-term costs, both in terms of the work, and in the perceived opportunity cost of deferring features. But by working to help the Product Owner to prioritize "technical stories," you are realizing the Scrum principle of *Transparency*, and building trust between the Product Owner and the Development Team.

Even if your project dynamics require you to address infrastructure in some other, more indirect, way, you can benefit by starting to think in terms of how what you want to do adds value. Software engineering is a means towards the end of solving business problems, and as engineers we should be able to explain what we're doing to any informed stakeholder.

This is an updated version of an article posted on Accidental Simplicity in June of 2009.

Agile Scrum