Agile Product & Project Management Advisory Service
Executive Update Vol. 11, No. 21

Starting Agile Adoption:

Part II — Avoiding Common Pitfalls of Planning

by Steve Berczuk

Agile software development involves people working together, across disciplines, to deliver business value efficiently. While the Agile Manifesto states that agile development values "responding to change over following a plan" and "working software over documentation," that does not mean plans are not important. A plan allows you to measure your progress, focus your efforts, or, more important, present a target that stakeholders can invest in. Too much time planning is wasteful, and it can be tricky to balance the lean imperative for reducing waste, with the comfort that a complete plan up front gives you. Collaboration makes agile effective and allows you to plan more efficiently and keep the focus on delivering value. This *Executive Update*, the second in a three-part series about starting agile adoption, discusses how to avoid some of the common pitfalls of adopting agile planning.

OVERVIEW OF AGILE PLANNING

Because agile methods downplay the role of up-front documentation, a common misconception is that agile planning is undisciplined, favoring an approach that builds something now and fixes it later. In practice, planning is essential to successful agile software development.

Agile planning uses lightweight artifacts that enable teams to capture the essential aspects of requirements quickly and with minimal cost. Your plan enables you to identify problems and adjust to address new issues and market demands. With an agile planning process, the team can identify high-cost, high-risk items quickly as well as move forward on high-priority and well-understood items without excessive overhead.

Successful agile planning requires significant discipline and frequent, honest feedback. Agile plans are:

- **Incremental.** Each unit of planning demonstrates some increment of functionality that you expect can be completed in the sprint.
- **Measurable.** Each increment of work should have a definition of completeness that all stakeholders agree on. This helps make progress more measurable by providing the team and stakeholders a point at which to make a binary decision about a story's completeness, without having to judge percent complete, which is often not actionable.
- **Iterative.** At the end of a sprint, stakeholders review the current state of the working software and compare progress to the plan. You evaluate project risk at the end of each iteration, rather than at the end of the project, and you base your evaluation on the state of working software. Sprint reviews are essential for developing an effective agile planning process.
- **Collaborative.** Software development is complex, and requirements and estimates have context and assumptions. Involving the right people in planning discussions early helps to significantly mitigate the project risks.
- **Focused on business value.** Everything on the backlog, whether it originated from a customer request, a product owner's vision, or an engineering need, should be prioritized in terms of the value that the item delivers to the organization.

At the center of agile planning is collaboration. In *Lean Architecture*, Jim Coplien describes the lean secret: "Everyone All Together, From Early On."[1] By providing a framework to identify risks and issues collaboratively, agile planning techniques aim to get the feedback you need to plan effectively.

PLANNING AND THE PLAN

An agile approach mitigates risk by having frequent, high-quality communication among all stakeholders. This includes developers, testers, product managers, and other stakeholders who can provide insight into

costs and tradeoffs as well as those who can answer questions and clarify misunderstandings. The high degree of participation and collaboration can help your team be more productive.

No matter how complete a specification appears, for all but the most trivial features, the context provides insight into the intention of a requirement. More detailed written specifications can give you a sense of security and a feeling that you've understood the problem fully, but often detailed specifications are not completely read, expensive to keep up to date, and become quickly outdated as your team adapts to an obstacle. There will always be uncertainty. By erring on the side of less up-front documentation in favor of more interaction, including demonstrations of running code, you can make the most of the effort spent on planning and start evaluating working systems sooner.

USER STORIES AND USE CASES

The first step in an agile planning process is a user story.[2] A user story is a lightweight starting point for understanding priority, cost, and for exploring alternative solutions.

A useful template for a user story is "As a <User> I want to <do something> <so that>."[3] This template covers the essential elements of a user story:

- The user who will benefit from the story
- The task the user wants to perform
- The benefit providing this functionality will provide

While this template may feel contrived in some cases, it's rare that you can claim to really understand a need without having expressed all of these elements.

User stories don't replace more detailed analysis techniques, such as use cases, for any but the most trivial requirements. A user story provides you with a way to quickly identify business value risk, as well as which stories could benefit from more detailed analysis and for which of these where the cost of the analysis would not be commensurate with the value.

A user story approach also encourages a focus on delivering business value and helps teams move away from a "show me the spec" attitude to a more collaborative one.

TECHNICAL REQUIREMENTS AND USER STORIES

Teams transitioning to agile struggle with how to manage technical and infrastructure tasks on their project backlog while still allowing business value to be the primary driver of prioritization. Such items as setting up testing infrastructures and repaying technical debt are important to a project's success, but they are often considered outside of the realm of the product owner's realm of prioritization.

A technical task can and should also be expressed in the same terms as feature-oriented user stories. To be successful, agile software development depends on technical practices. But business priorities should determine what you do. You can balance this by viewing tasks that seem like nonfeature work through a product-value lens:

- Include such tasks as refactoring or repaying technical debt in the estimates for the story in which you will do the work.
- Explain infrastructure tasks that don't connect directly to features in terms of product value.

It's important for the development team to work with business goals in mind. Agile projects are a collaboration between technical and businesspeople, and the business should be able to understand the value that a technical improvement brings. In the end, success of this process depends on the product owners and development team establishing trust and a common set of goals.

PRIORITIES COME FIRST

Given a set of features in the form of user stories, the next step is to prioritize.

A product owner asks a team to estimate a large number of backlog items so that he can prioritize them, taking cost into account. The rationale is often "feature A is only important if it's not expensive." There are three problems with this approach:

The *Executive Update* is a publication of the Agile Product & Project Management Advisory Service. Print ISSN: 1946-7338 (*Executive Report*, *Executive Summary*, and *Executive Update*); online/electronic ISSN: 1554-706X.

1. It requires the team to spend significantly more time in estimation and planning activities than if the product owner had prioritized first.
2. It takes away an opportunity to rethink a feature that seems important but costly. While it might not make business sense to implement one feature when it is as expensive as three others combined, there may be a light version of the feature that the team and the product owner can define that will address the core needs.
3. By making estimated cost an early input into the prioritization process, the product owner is ceding control of the backlog to the team.

It still may be true that in the end, a feature is too expensive to implement, but in many cases, a team can make tradeoffs or reshuffle the priority. Time spent estimating is time not spent developing features. While estimation and planning are essential activities, they provide the most value when connected to features that are likely to implemented.

ESTIMATION

Planning Poker, as described in Cohn's *Agile Estimating and Planning*,[4] is a useful agile estimation technique. In Planning Poker, the team members simultaneously and independently estimate user stories, often using cards with estimates printed on them. Everyone on the team independently provides estimates, and the range of estimates gives you insight into the risk in developing a feature.

The advantages of Planning Poker over more detailed or formal estimation approaches are that it is:

- **Collaborative.** You quickly gain insight into the team's level of confidence in developing a feature.
- **Interactive.** You avoid groupthink, since all estimates are placed on the table at once.
- **Quick.** You quickly identify features that require additional discussion.

Some common pitfalls teams encounter are:

- **Too much discussion before an initial estimate.** A planning session can become very long if every story is discussed in detail. The need to do this discussion is an indication that the stories are too vague to estimate. Stories that are too vague to estimate are often hard to test or quantify in terms of business value.
- **Having only a subset of the team in the room.** Trying to optimize people's time by having a select group in estimation sessions is also risky. While it may seem like you are saving time, software systems are so complex that it is difficult for any one person to understand the implications and possibilities of a feature. Having the entire team in the room allows you to evaluate risks and alternatives more accurately.

Here is an effective approach that both speeds up the process and identifies uncertainty:

- **Review the story.** The product owner should briefly describe the feature and its value to the user. This should not be a long presentation. If the story has inconsistencies or is too vague, this will become clear during estimation. You want to cover the entire backlog quickly and focus discussion based on what you discover during the estimation phase.
- **Estimate.** All should estimate, even if they are uncertain of the work involved. Over time, the team will collect data and develop a feel for how much effort a certain type of feature takes to estimate.
- **Compare and discuss estimates.** If you compare estimates, and the estimates are similar, ask one person to describe the assumptions in terms of details and implementation. If the team and the product owner agree, you have some level of confidence in the estimate. If the estimates vary widely, or there is disagreement about the basis for the common estimate, you have a high-risk story that needs further analysis.

Remember that agile software development is a collaborative activity and that you lose the benefits of collaboration by limiting who participates in planning and estimation.

If your planning and estimation sessions seem long and unproductive, consider doing some preliminary work among the product owner, ScrumMaster, and QA team to make sure your stories have the right level of fidelity and are at a reasonable level of granularity before including the whole team. As we learned in Part I, this can mean combining or splitting stories or having the product owner work with the QA team (or the ScrumMaster) to refine the story.[5]

EFFECTIVE MEETINGS

Underlying all these techniques is the idea of effective meetings. Whenever you get a group of people together, there is potential to feel like you are wasting time. As a consequence, managers often try to optimize meetings by having fewer meetings and being selective about

whom to invite. This often leads to more meetings later, after the team has been struggling unnecessarily.

Jean Tabaka discusses some good techniques for agile team meetings in *Collaboration Explained*.[6] Collaboration doesn't happen in a vacuum, so try to provide frameworks where team members can make the best use of their time.

CONCLUSION

Agile planning is simple in concept but hard to do, and it's easy to mistake its lightweight aspect for lack of discipline and detail. In agile planning, you try to apply your energy efficiently by doing more detailed analysis only for stories that have more complexity.

To be successful at agile planning:

- Include team members from every aspect of the project in planning and estimation. This will help you to quickly identify alternatives and avoid making incorrect assumptions.
- Prioritize stories before spending time estimating them. The business value should drive what you want to do. If estimates seem out of line, you can revisit.
- Express everything on the backlog in terms of business value, including technical tasks.
- Expect that it will take a couple of iterations for the planning process to go smoothly.

Agile planning is iterative, both in terms of the process and how the process evolves. Apply the idea of iteration and review to the planning process itself to tune the process to suit your organization. While this *Update* describes a single agile planning cycle as a linear process, on projects you will have multiple planning cycles that use the results of — and the things you learned from — prior sprints to adapt, focus, and improve. In *Agile Retrospectives*, Esther Derby and Diana Larsen provide excellent guidance about how to do effective retrospectives.[7]

By having the right people in the room and keeping the process lightweight, you can deliver useful software more quickly with less planning overhead. Part III will discuss issues around developer or unit testing on an agile team.

ENDNOTES

[1]Coplien, James O. *Lean Architecture: for Agile Software Development*. Wiley, 2010.

[2]For more on user stories, see: Cohn, Mike. *User Stories Applied: For Agile Software Development*. Addison-Wesley Professional, 2004.

[3]Cohn. See 2.

[4]Cohn, Mike. *Agile Estimating and Planning*. Prentice Hall, 2010.

[5]Berczuk, Steve. "Starting Agile Adoption: Part I — Quality Assurance." Cutter Consortium Agile Product & Project Management *Executive Update*, Vol. 11, No. 16, 2010.

[6]Tabaka, Jean. *Collaboration Explained: Facilitation Skills for Software Project Leaders*. Addison-Wesley Professional, 2006.

[7]Derby, Esther, and Diana Larsen. *Agile Retrospectives: Making Good Teams Great*. Pragmatic Bookshelf, 2006.

ABOUT THE AUTHOR

Steve Berczuk is an engineer and ScrumMaster at Humedica, where he's helping to build next-generation clinical informatics applications based on software as a service (SaaS). The author of *Software Configuration Management Patterns: Effective Teamwork, Practical Integration*, he is a recognized expert in software configuration management and agile software development. Mr. Berczuk is passionate about helping teams work effectively to produce quality software. He has a master's degree in operations research from Stanford University, a bachelor's degree in electrical engineering from MIT, and is a Certified Practicing ScrumMaster. He can be reached at steve@berczuk.com.

