How Agile Practices Reduce Requirements Risk

by Ellen Gottesdiener

Every software project carries some risk, but many of these risks can be mitigated. That's true of problems related to product requirements—problems that are often cited as one of highest risks for any type of software project. Whether it is having unclear requirements, lack of customer involvement in requirements development, or defective requirements, these troubles are a major culprit in projects that go awry.

Project teams can make a difference by adopting and implementing agile practices. When implemented correctly, agile practices greatly mitigate the most common risks associated with requirements on software development projects. Adapting the requirements risks I discuss in my book *The Software Requirements Memory Jogger* [1], I will explain how agile practices act to mitigate the risks—and, therefore, provide the business value for which these practices aim.

Risk 1: Unrealistic Customer Expectations and Developer Gold-Plating

This is the risk that your customer's wishes will exceed what your team can deliver or that developers—in their sincere quest to satisfy their customers—will add unnecessary features.

How does agile address these problems? In agile projects, we chop up delivery expectations into short iterations—one- to three-week timeboxes. Each timebox begins with an iteration planning workshop in which the customer decides which work should be delivered.

The process is entirely transparent:

1. The customer states the goal or theme for the iteration.
2. The delivery team members state how much time they have to devote to the effort (i.e., their capacity, usually in work hours).
3. The customer selects the highest-priority requirements from the backlog (the master catalog of work needed to build the product).
4. The desired requirements are further discussed and elaborated on, as needed.
5. The team estimates and tasks out the work.
6. The team and the customer explore risks and dependences.
7. The team makes an explicit commitment about which requirements will be delivered.

As an analyst and coach, I find that the key to this process is having each work item (also called a *story*) small and sharply defined. If you don't know the completion criteria up front—to assess whether a requirement is "done"—then the customer's expectations might be dashed or delivery team members might make (wrong) assumptions and add extras.

Throughout the iteration, the team checks on expectations by showing completed stories to the customer. At the completion of each iteration, team members show any stakeholders all the completed work in a demonstration and review.

Risk 2: Insufficient Customer Involvement

The most commonly cited project risk is a lack of engagement by customers. A precondition on agile projects is that we require the customer to participate throughout each iteration. As just described, the customer declares the iteration goal and work at the start, reviews completed work during the iteration, and attends the demo or review at the close of every iteration. In addition, the customer must always be available to answer questions about requirements.

When customers are less available, then domain-knowledgeable business analysts act as proxy customers to help with requirements analysis. Thus, business analysts become customers; they are delegated the decision-making authority

ISTOCKPHOTO

about requirements priorities. In other cases, I have seen business analysts act as coaches and aides to customers, helping them define concise and clear requirements, prune the ever-changing product backlog, analyze backlog items to prepare the team for the iteration planning workshop, and document requirements.

No matter who assumes the customer role, it is front and center on an agile project.

Senior-level customer involvement is also crucial, particularly on large, complex projects. These executives set the context for the product development effort by participating in product roadmapping to define the product vision and lay out which features will be delivered over time based on market and technology needs and constraints.

Risk 3: Poor Impact Analysis

It is rare to encounter products with fixed, clear requirements up front. Changes to requirements and shifting priorities can affect the sequence of work, introduce unforeseen rework, or create product defects.

Poor impact analysis involves not understanding the ways that new and evolving requirements affect the set of proposed requirements that make up the *baseline* (the traditional requirements term) or *backlog* (the agile term).

On agile projects, it's OK to change the backlog. Indeed, some teams agree that any team member can revise the backlog at any time. Other teams allow only the customer or the business analyst to modify the backlog. Whichever arrangement the team agrees to, the point is that team members recognize that the backlog of work is dynamic.

The product backlog is continually analyzed and adjusted. The customer, often with an analyst and perhaps other team members, *prunes* the backlog. When pruning, impact analysis is key: Items are broken down, analyzed for their interdependences, shifted up or down in priority, re-estimated, removed, and reallocated to iterations or releases. This happens weekly on most agile teams. Analyzing the impact of changing requirements is part of the rhythm of successful agile teams.

Risk 4: Scope Creep

The uncontrolled expansion of requirements throughout the project is the highest risk of any software project [2]. In addition, the larger the product, the more requirements grow. Yet, scope creep might actually be considered "normal."

Most software products present a wicked dilemma: The problem you are trying to solve is not fully understood until *after* it has been solved (i.e., some of the solution space lies within the problem space) [3]. If you cannot know what the solution is until you start to build the product, you benefit by starting to build it in small increments and then obtaining feedback to learn and adapt. This is the essence of agile development.

Some stability is, of course, necessary. Product goals, objectives, target market and users, and a product vision need to be articulated (agile teams do this as part of product and release planning).

It's OK to add new items or stories to the backlog as they arise. Agile teams manage scope creep by continually pruning the backlog.

The project's scope is defined at a high level but is not a binding contract. By working in short delivery cycles on a small subset of requirements, agile teams can better control scope. Every one to three months, they conduct release planning to adapt the requirements delivery plan over a longer time frame.

Risk 5: Defective Requirements

Requirements defects include missing, erroneous, conflicting, or ambiguous requirements, which can lead to a defective product. Even worse, it can lead to building the wrong product. On an agile project, small, concise requirements (stories) are sharply defined once the customer has chosen them from the backlog. Defining story "doneness" is essential. As mentioned earlier, the customer participates in iteration planning and is available throughout each iteration to answer requirements-related questions.

That leaves no wait time during which developers or testers make (wrong) assumptions about requirements. In addition, I like to have the team develop user acceptance tests as soon as work begins on each item. This form of validation is the best way to remove ambiguity from requirements.

A requirements defect also leads to excessive rework—revised code, additional testing, modified documentation, and premature or unnecessary analysis. On agile projects, we do not analyze backlog items until they move to the top of the backlog stack—when they are about to be pulled into an iteration planning workshop. This practice not only prevents us from analyzing requirements that will never be implemented but it also avoids rework caused by analyzing requirements prematurely. Additionally, many requirements are interdependent. When you analyze, build, test, and deliver a requirement, you learn things that will impact your understanding of related requirements. By waiting until the "last responsible moment" to conduct analysis, you are better informed and can tackle your analysis more efficiently.

Risk 6: New Processes and Tools

How do agile teams reduce the risks associated with using new requirements practices and tools? How do teams mitigate the normal risks of *any* change? They minimize these risks through feedback, metrics, and coaching.

Each day, the team shares feedback via a *standup meeting*. In that fifteen minutes, team members state what they did yesterday, what they plan to do today, and what (if any) impediments they are experiencing. The team also gets customer feedback by showing its customers completed stories as soon as they are finished. The other key feedback mechanism is iteration *retrospectives*, sessions during which team members review self-correcting feedback and identify small, focused adjustments that will help them better integrate changing work practices.

A key metric for agile teams is the burn down chart, showing the rate at which stories or tasks are being completed, measured in hours per day. See this issue's "Getting the Most Out of Burn Charts" for more information.

Real Risk Reduction

Myths abound about how agile practices ignore or avoid good requirements practices and can increase requirements risks. In reality, agile done right decreases common requirements-related risks. Adapting agile practices can enable the team to act in rhythm with the dynamic nature of requirements development and facilitate the delivery of "solutions that meet business needs, goals, or objectives" [4]. **{end}**

References

[1] Gottesdiener, Ellen. *The Software Requirements Memory Jogger: A Pocket Guide to Help Software and Business Teams Develop and Manage Requirements*. GOAL/QPC, 2005.

[2] Jones, Capers. "Social and Technical Reasons for Software Project Failure." CrossTalk, June 2006, pp. 4-9.

[3] DeGrace, Peter and Leslie Hulet Stahl. *Wicked Problems, Righteous Solutions: A Catalogue of Modern Software Engineering Paradigms*. PTR Prentice Hall, 1990.

[4] International Institute of Business Analysis. *A Guide to the Business Analysis Body of Knowledge*, version 2.0, 2009.

How do you balance agile's imperative to define small, concise requirements in each iteration with the need to have a larger view of the entire product's requirements?

Follow the link on the **StickyMinds.com** homepage to join the conversation.