Agile Requirements: Not an Oxymoron

by Ellen Gottesdiener

Adult children. Jumbo shrimp. Seriously funny. I'm sure you recognize these expressions as oxymorons—self-contradictory phrases, often with an ironic meaning.

Should we add "Agile requirements" to the list? Does Agile development fit in with traditional requirements practices? And if so, how?

Once More into the Breach

Traditionally, defining requirements involves careful analysis and documentation and checking and rechecking for understanding. It's a disciplined approach backed by documentation, including models and specifications. For many organizations, this means weeks or months of analysis, minimal cross-team collaboration, and reams of documentation.

In contrast, Agile practices—Lean (http://www.leanprimer.com/downloads/lean_primer.pdf), Scrum (http://www.scrumalliance.org/), XP (http://www.extremeprogramming.org/), FDD (http://www.featuredrivendevelopment.com/), Crystal (http://www.amazon.com/exec/obidos/ASIN/0201699478), and so on—involve understanding small slices of requirements and developing them with an eye toward using tests as truth. You confirm customers' needs by showing them delivered snippets of software.

However, Agile projects still produce requirements and documentation, and they involve plenty of analysis. On the best Agile projects, requirements practices combine discipline, rigor, and analysis with speed, adaptation, and collaboration. Because software development is a knotty "wicked problem" with evolving requirements, using iterative and Agile practices is not only common sense but also economically desirable.

Indeed, Agile requirements drive identifying and delivering value during Agile planning, development, and delivery.

Planning

Agile teams base product requirements on their business value—for example, boosting revenue, cutting costs, improving services, complying with regulatory constraints, and meeting market goals. If you're agile, it means that you focus on value and jettison anything in the product or process that's not valuable.

Planning covers not only the "now-view" (the current iteration) but also the "pre-view" (the release) and the "big-view" (the vision and product roadmap), with close attention to nonfunctional (http://www.agile2010.org/express.html#5210) as well as functional requirements. The product roadmap is crucial for keeping your eyes on the prize, especially in large, complex products. You don't have to know each specific route, but the overall way must be clear. It's driven by the product vision and marked by industry events, dates, or key features that must be achieved along the route.

Customers (or "product owners," in Scrum terminology) drive Agile planning, constantly reprioritizing requirements and evaluating risks and dependencies. Close customer collaboration is essential. One of the original Agile methods, DSDM (http://www.dsdm.org/), has customer involvement as the first principle.

Your Agile backlog, or catalog, of product needs changes constantly—whenever you do planning (e.g., for a release or iteration)

or, if you're using a kanban/flow model, every time you're ready to pull in another requirement. Plans are based on deciding what to build, and when.

An Agile delivery team works ahead, preparing requirements for development and testing. This preparation is vital to deliver the value as soon as possible, with smooth flow and no thrashing or interruptions in delivery and testing.

Developing

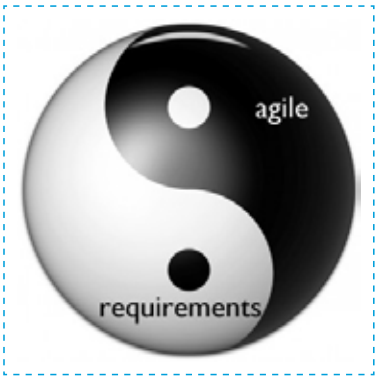
An Agile team's work is based on building concise, fine-grained requirements (typically captured as user stories). Developers need small, tamped-down requirements to work from. Small requirements that have clear conditions of satisfaction (doneness) minimize risk.

The team may also sketch organic data models, state diagrams, and interface mockups. These are like micro-specifications: "ready" requirements for pulling into delivery. The team knows enough to estimate, develop, test, and demonstrate the requirements.

Doneness is a key aspect of requirements. I wrote about "done" requirements in my first book (2002, http://www.ebgconsulting.com/Pubs/reqtcoll.php): the team and customer need to know when they understand the requirements enough to build and test. This concept is used often in Agile development and refers not only to requirements but also to the build, test, and release process.

Delivering

Requirements are built and released based on the team's clear understanding of requirements dependencies, which also drive architecture trade-off decisions. Requirements are dependent on each other when each relies on (and thus constrains) the other.



Smart Agile teams analyze development and delivery dependencies (http://www.agile2010.org/express.html#5364) to optimize value. Traditional requirements models are useful for dependency analysis and to supplement Agile's lightweight requirements (such as user stories).

It's All Good

"Agile requirements" isn't an oxymoron, although it may be a bit of a paradox—in the same way that the concise enables the complex, the small gives rise to the large, incompleteness facilitates the finish, and you must slow down to speed up. Indeed, Agile requirements are central to Agile planning, development, and delivery.

> About the author

Ellen Gottesdiener

EBG Consulting, Inc. Principal Consultant and Founder Ellen Gottesdiener helps business and technical teams collaborate to define and deliver products customers value and need. Ellen is an internationally recognized facilitator, trainer, speaker, and expert on requirements development and management, Agile business analysis, product chartering and roadmapping, retrospectives, and collaborative workshops.

Author of two acclaimed books Requirements by Collaboration and The Software Requirements Memory Jogger, Ellen works with global clients and speaks at industry conferences. She is co-authoring a book on agile practices for discovering and exploring product needs. View her articles, tweets, blog, and free eNewsletter on EBG's web site.