Product Arts

Agile Requirements – So What's Different?

Part 3 of the Product Requirements in a Nutshell Series

Parts 1 & 2 discuss the Requirements Process from traditional (waterfall) point of view and which focuses on documenting the features and functionality of the system to be built in advance of building the solution.   The artifacts for this are the requirements docs and specs. At the complete other end of the requirements spectrum is the software development methodology of Agile/Scrum.   The main thrust of this method is to AVOID the creation of the formal requirements documents and to use the actual product as the spec.   This article explores the methodology and compares whether the common problems identified in Part 2 for traditional techniques are overcome by Agile.

For more background, see Part 1 – A Tour of Requirements Documents, and Part 2 – 4 Common Requirements Issues.

Agile Background

Agile originated from the custom software development world where gathering requirements from stakeholders and managing changing requirements proved to be the major project challenges.   The process has evolved into a major commercial software process.   The Agile methodology has been coupled with Scrum which focuses on specific project management and development flow.   The main thrust of Agile/Scrum comes from its Manifesto that focuses on:

- Individuals & interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan (adaptive versus predictive)

The Scrum process utilizes short development cycles (often 2-4 weeks) called Sprints with very limited scope to produce a functional and potentially releasable set of features.   At the completion of the Sprint, the software can be tested by actual users for validation.   A typical release has multiple Sprints to deliver the full set of envisioned functionality iteratively.

The concept of delivering product to the world in an incremental or iterative way is not new and has existed for over 20 years.   An early iterative software methodology was the Spiral Method that focused on delivering increasingly functional prototypes that validated the product concept in sequential iterations.   Initial requirements would be generated, built and validated and then the requirements would be revised and extended for the next iteration.   The main purpose of the model was for highly risky projects that could be aborted at any point when it became apparent the project objectives could not be met.

Another popular iterative model is called the Rational Unified Process (RUP), originally developed by Rational Software (now IBM).   RUP relies on traditional MRD/PRD/FSD docs to pre-plan the entire set of deliverables and then to deliver the functionality in pieces called Iterations.   Whereas Spiral evolves the requirements prior to each iteration, RUP develops them at the beginning of the project.

In contrast, Agile/Scrum develops the requirements on the fly during each Sprint. A primary driver for this comes from the belief that the entire set of requirements cannot be known ahead of time, as users often cannot state what they want or need until they see it.

Prior to each release, a set of User Stories are developed for the list of target features as part of the Product Vision process. These User Stories are very simple statements of User Goals or Actions to be accomplished and written on note-cards, or into spreadsheets or requirements software tools. An example User Story for an online travel booking site could be *"As a business traveler, I would like to book a hotel near my meeting location for convenience."* The target User Stores are estimated for effort at a very coarse level prior to the Sprint and rank order prioritized into a Feature Backlog.

Each sprint then targets a small number of the User Stories to implement from the Backlog. Within the Sprint, the User Stories are refined <u>in near real-time</u> as developers interact directly with product managers or customer representatives to identify the user interaction steps, qualities and constraints on the system, and define test cases. User interface questions and decisions can also occur on the fly. There are no required MRD, PRD, or FSD and the working software is the formal documentation. In some environments, the developed requirements may go back into a formal document, simply for tracking and compliance purposes, but not for planning.

Comparing Agile to Traditional Requirements

Similar to the requirements document mapping we used in Part 1, we can also map our original classifications to the Agile/Scrum process that looks like the following:

	"MARKET & BUSINESS DRIVERS"	**"USING THE PRODUCT"**	**"WHAT WE NEED TO BUILD"**
Traditional Methods	Market Requirements Document (MRD)	Product Requirements Document (PRD)	Functional Specification Document (FSD)
Agile/Scrum	Product Vision Doc	Iterated User Stories	The actual code

The concept of a Product Vision Document exists to identify the Market & Business Drivers similar to the traditional process MRD and to define the intended scope of the release. Iterated User Stories per Sprint have replaced the detailed PRD, and the delivered code eliminates the need for the FSD. The result is usually much faster time to market with a subset of the eventual desired functionality and is purported to result in happier customers and developers.

If we look at our common requirements issues identified in Part 2 for traditional methods, we can see where Agile/Scrum stacks up. For <u>*Issue #1 - Missing Market & Business Drivers*</u>, Agile can suffer from the same problems as traditional software in not understanding what you're trying to accomplish and why. While the Agile process may appear to be able to reveal MUCH SOONER that you're spinning your wheels, this is not really a good reason to use it. Without a good vision for the market problem and measurable business objectives, it becomes nearly impossible to effectively prioritize the feature set for any development methodology. Like the saying goes "if you don't know where you're going, any road will get you there". There are cheaper and faster ways than Agile to figure out first what you want to do. See the discussion on <u>*Issue #4 - Market Validation*</u> below.

For *Issue #2 – Focusing on Features, not on Accomplishing User Goals*, this is where Agile User Stories really shine and force the discussion around user value as opposed to solution features. The process of developing User Stories first starts with identifying User Roles or User Personas that define their overall goals and activities related to your product. The format of a User Story is actually more of an ideal outcome that maps to the user's overall goals, "As a <User Role>, I want to <achieve task or goal> because <benefit it provides>. This inherently shifts the focus away from Features. Note that the concept of User Stories can be applied in traditional requirements through Usage Scenarios, so this need not be the exclusive domain of Agile.

For *Issue #3 – Missing or Poorly-Defined Qualities & Constraints*, Agile can suffer from the same problem as traditional methods, in that Qualities and Constraints can be completely missed or ill-defined. Some issues may be dealt with faster in Agile, such as Usability, however some may be more easily deferred and forgotten due to the iterative discussions focusing on functionality. Issues such as Scalability, Data Security, and (eventual) Internationalization can be completely missed if not somehow identified in a User Story, forcing a major refactoring to occur in the architecture somewhere down the road. In this situation, it's a "Pay me Now or Pay me Later" choice, with Agile generally slanted towards "Pay me Later" after you've proven you can get users with the basic value proposition. For most consumer products, this is probably a fine approach, but for more critical enterprise applications this oversight can delay a real deployment and negate some of the value of the Agile methodology in getting something to market faster.

Finally, for *Issue #4 – Lack of Market Validation*, Agile can also have a bad start. While the primary driver for Agile is to get a working product to market for testing as quickly as possible, the product can then languish for several release cycles trying to find a real problem to solve. Therefore, as indicated above for Issue #1, it's worthwhile to do a simple mockup in the Product Vision stage before wasting development time, even in Agile. It's going to be much faster (and cheaper) to validate that your basic value proposition is going to fly in this manner before spending 3-6 months trying to hammer out both a product and its value.

Summary

Agile can suffer from some of the same issues that plague traditional requirements efforts, especially around defining the market problem and business objectives to be achieved. The Agile process can also suffer from inattention to Qualities and Constraints in the requirements, though the impact may be less than in traditional methods since a new release is just around the corner.

Where Agile shines is in focusing on User Goals instead of Features through the application of User Stories and similar concepts can be applied using traditional methods. This, coupled with shorter development cycles, enables the product to more quickly zoom into creating real user value. This adaptive approach aligns exceedingly well with web-based software. Note that this result is not impossible to achieve using traditional methods, but does require a level of diligence in prototyping and market validation not typically found in the traditional environment.

About Product Arts

Product Arts specializes in Product Management consulting and training. Our training includes public and private custom training on Product Requirements. For more information, go to www.product-arts.com or email info@product-arts.com.