Gartner Research

Publication Date: 7 July 2011 **ID Number: G00213381**

Agile and the Rest of the Organization: Requirements Definition and Management

Thomas E. Murphy

Requirement defects are the leading source of defective applications being delivered to users, which results in project failures and cost overruns. Although agile practices provide a quicker feedback loop, they also apply new approaches to requirements and create challenges as organizations look to move to enterprise agile.

Key Findings

- Agile methods focus on collaboration — well-fitting tools and practices enable this to scale across the IT organization.
- Regardless of method, the leading source of defects delivered to users is requirements defects.
- Too many companies still rely on spreadsheets and text documents for requirements.
- A growing number of tools enabling improved requirements definition are entering the market.

Recommendations

- Use approaches that enable incremental capture and delivery of requirements and function.
- Create requirements in a modular format that can be readily consumed as agile tasks.
- Incorporate a broader set of requirements techniques that will enhance communication, such as models and wire frames.
- Recognize the commitment to collaboration that's required from all participants.
- Ensure that nonfunctional criteria are adequately addressed in standards and practices.
- The selection of agile methods and practices for a given project should be based on a variety of factors, which may dictate something other than agile.

ANALYSIS

The Agile Antipattern

The "Agile Manifesto" states, "we have come to value working software over comprehensive documentation." This statement seems often to be taken to mean, "If my code 'works,' then I don't need documentation." Combined with the fact that most agile methods don't specify a "requirements process," this can lead to confusion and cause roadblocks for the enterprise use of agile methods. However, this is not to say that agile is more broken with respect to requirements. Requirements are the leading source of delivered defects for the average development team (see Capers Jones, "Software Quality in 2010, A Survey of State of the Art"). Across the market, we fail at capturing and managing requirements in an effective fashion. However, agile, especially when applied to enterprise and/or distributed projects, creates additional stress on the system.

Agile techniques focus on the idea of collaboration and continuous feedback. Requirements tend to evolve and come together over time. Evolution of requirements means adapting to user needs, but it also creates instability in the project when the rest of the team desires stability. In some cases, a pure-agile approach may not be appropriate, due to project constraints (see "Enterprise-Class Agile Development Defined").

The Current Norm

Many organizations practice a hybrid of structured and agile practices. Projects start with business-level requirements captured in traditional requirement specification documents. These provide a foundation for planning — i,e., how much and how long — and create a "fixed schedule" baseline that will be used for the overall project. Depending on the format of these requirements, once the project moves forward, the development team will break them down and load them into their product backlog as user stories and begin agile development (i.e., the application is built over a number of iterations).

This initial collection also enables the testing team to create a test plan. When all the development work is complete, the code moves into testing and delivery, and, once again, it follows a traditional, structured process. This introduces a number of inefficiencies, but organizations often want a control process on either end of the project and feel that pure agile may not meet their compliance needs.

However, we believe that, done correctly, there is much to gain by effectively executing across the application life cycle using a more-agile approach. As a starting point, shifting to a more modern requirements process can produce fewer errors and a better connection to validation. This should include looking at products that provide a broad variety of techniques, including wire frames or prototypes, use cases or use stories, business flow diagrams or process models. This could be realized by using different tools or a single environment. In all cases, these tools take a more discrete view of requirements, rather than the traditional functional specification. Start with requirements that are more componentized, and they will be easier to track, easier to cost and be more conducive to agile planning methods, such as scrum and kanban.

What's the Story?

Agile practices generally focus the requirements process around the concept of stories and/or use cases. There are a number of key strengths to a story approach:

- A focus on objectives
- Easy connection to "backlog" for planning and prioritization

- An incremental approach
- A short and simple method

Extreme programming (XP) introduced the concept of story cards and the planning game. Simple 3x5 cards with a title and a description. Scrum teams often use sticky notes to capture stories and carry out planning on the whiteboard. However, Post-it notes and the stories themselves aren't the complete picture. And if requirements are not captured into software, it becomes difficult to have traceability and to meet compliance processes for audits. In addition, a user story is really just designed as a placeholder for a conversation. If the conversations don't happen, and if the outcomes or decisions are not recorded, then the process falls apart.

Use cases provide more formality, along with the ability to develop the requirements incrementally. This has been well-outlined in the book "Writing Effective Use Cases" by Alistair Cockburn. In its initial form, a use case is similar to a user story, but use cases come with the semantics to add details, such as those required to define the user acceptance test (UAT) criteria.

As agile practices evolve from tactical adoption to strategic use, the elements of agile requirements also evolve. This is partly to deal with consistency through the definition of personas and the capture of nonfunctional requirements, but also through defining a better view of a long-term plan captured in a sequence of:

- **Investment themes —** top-level objectives that may span projects and products
- **Epics —** groups of related user stories
- **User stories —** an independent, negotiable, valuable, small, testable requirement that can be estimated
- **Tasks —** the work items that will implement stories

These topics are detailed in "Agile Software Requirements" by Dean Leffingwell, and we are starting to see these approaches being implemented by agile application life cycle management (ALM) tool vendors (e.g., Rally Software, Atlassian, VersionOne, CollabNet and Thoughtworks Studios). More important than the tools is first getting the practices and training in place. Then the tools can be used to enable the chosen practices.

Use the Right Tool

Although story formats provide a good way to define requirements, they're not the complete picture. Pictures are another valuable format, and a number of tools have entered the market that enable teams to create prototypes or wire frames. We find them to be valuable formats for communicating with the users of the system, because the users often can't express what they want until they can see it. The resulting defects tend to make it through from requirements definition to the UAT, when the team learns that "what we really wanted was ..." at a point where the cost will be much greater.

Depending on the implementation tools, developers may prefer to build a quick and dirty prototype. The advantage of focused tools is that they normally also include facilities to capture comments and decisions. As companies shift from small projects and teams engaged in agile to more-complex projects and potentially distributed teams, there is a need to shift from paper and spreadsheets to tools that provide workflow, persistence and traceability. At the same time, caution must be taken to avoid stifling teams' ability to establish a productive flow.

Behavior-Driven Development

A newer techniques in agile development is behavior-driven development (BDD), which is an extension of test-driven development (TDD). Unit tests serve for many teams as part of the documentation, as well as a facility to aid in designing the implementation. BDD is designed to enable business users, testers and developers to collaborate on the goals of a project and the features that will be implemented.

Generally, the practice makes use of a domain-specific language to capture these goals and behaviors such that acceptance tests can be automatically generated. This enables direct traceability from requirements to test cases and automates the creation of the acceptance tests from the requirements. BDD practices also use example creation as a form of prototypes to help drive the conversations. The best support for this development style is implemented in Ruby; however, tools are emerging in other languages as well. The concepts can be followed, and the base format of requirement definitions can be used in other requirements practices to focus on desired outcomes, rather than features:

- **Scenario —** Descriptions of desired behavior
- **Given —** Initiation states or preconditions
- **And —** Optional clauses
- **When —** Events that occur to start the scenario
- **Then —** One of more actions that will result

This practice also meshes well with the "Design by Contract" style developed by Bertrand Meyer. This approach will still take two to five years to mature; however, organizations that build a foundation of TDD and move to consistent terminology and formats for requirements will be positioned to take advantage of these frameworks.

Nonfunctional Requirements

One of the fears of agile development is that little or no attention will be paid to nonfunctional requirements. After all, where do you capture them? However, a lack of attention to nonfunctional requirements is a challenge in all development. In addition, many of these types of requirements (e.g., maintainability, portability, localizability, performance efficiency, security and scalability) should not be detailed at the project level, but should be part of corporate policies and designated practices. These requirements should be part of the continuous integration practices that are used by the project teams, including peer code reviews, static analysis and threat models.

The challenge of nonfunctional requirements is they can have a dramatic impact on downstream costs to maintenance of the application. They are often the source of what developers term "false positives," when static analysis tools are employed. Other ways that agile teams should be paying attention to nonfunctional requirements are through the use of design patterns. If your organization is skipping these practices, or doesn't capture or define standards for nonfunctional criteria, it is not an agile team, but an ad hoc development team that is ignoring technical debt and headed toward downstream headaches.

Take Action

A great many tools and practices are available to help organizations improve their requirements practices. These practices are designed to fit well with agile development teams and will support the scaling of agile practices to larger teams and a broader portion of the portfolio, as well as to more-regulated industries. However, don't expect tools to solve all issues. Shifting to agile is a cultural change (see "The Cultural Transition to Agile Methods: From 'Me' to 'We'") and to get from a more waterfall or iterative approach to pure agile also requires a shift for the business from a project focus to more of a product focus. Organizations must determine what degree of agile will work best; however, although tools won't make the transition automatic, improved requirement tools with appropriate training will improve all development processes. Organizations need to:

- Identify and define corporate standards for terminology and practices.
- Identify current issues in their requirements processes — e.g., defect root cause analysis.
- Ensure that their teams are adopting agile, rather than ad hoc, development processes.
- Choose requirements as part of an overall ALM approach, rather than in isolation.

Books and Resources

"Agile Software Requirements: Lean Requirements Practices for Teams, Programs, and the Enterprise" Dean Leffingwell

"Managing Software Requirements: A Use Case Approach" Dean Leffingwell and Don Widgrig

"Writing Effective Use Cases" Alistair Cockburn

"Agile Modeling" Scott Ambler

Sample Requirements Definition Products

- Serena Prototype Composer
- Blueprint Systems Requirements Center
- iRise
- IBM Rational Requirements Composer
- Axure
- Basalmiq
- Microsoft SketchFlow
- ForeUI
- ProtoShare
- Visure IRQA
- eDevTech

RECOMMENDED READING

Some documents may not be available as part of your current Gartner subscription.

"Beware the Emerging Role of the Project Analyst/Manager"

"Requirements Form the Foundation of Software Quality"

"Measure and Manage Your IT Debt"

"Effective Risk Management for Applications"

"The Three C's of Agile"

"Enterprise-Class Agile Development Defined"

"The Cultural Transition to Agile Methods: From 'Me' to 'We'"

REGIONAL HEADQUARTERS

Corporate Headquarters
56 Top Gallant Road
Stamford, CT 06902-7700
U.S.A.
+1 203 964 0096

European Headquarters
Tamesis
The Glanty
Egham
Surrey, TW20 9AW
UNITED KINGDOM
+44 1784 431611

Asia/Pacific Headquarters
Gartner Australasia Pty. Ltd.
Level 9, 141 Walker Street
North Sydney
New South Wales 2060
AUSTRALIA
+61 2 9459 4600

Japan Headquarters
Gartner Japan Ltd.
Aobadai Hills, 6F
7-7, Aobadai, 4-chome
Meguro-ku, Tokyo 153-0042
JAPAN
+81 3 3481 3670

Latin America Headquarters
Gartner do Brazil
Av. das Nações Unidas, 12551
9° andar—World Trade Center
04578-903—São Paulo SP
BRAZIL
+55 11 3443 1509