Adapting agile requirements practices to ongoing enterprise improvement efforts: DevOps to the rescue

Cherifa Mansoura, PhD

December 18, 2012

Although Development and Operations teams both face daily pressure for continuous improvement and shorter time to market, their respective priorities, needs, and methods can conflict. Cherifa Mansoura explains how to scale disciplined agile delivery and agile requirements practices in a DevOps environment for future maintenance and operations on existing products. She also discusses the role of a business analyst in ensuring good collaboration between the two teams.

Introduction

Using Disciplined Agile Delivery (DAD) approaches for requirements management of green-field projects is fairly straight-forward. In a green-field project, you know to begin with a Shared Vision and stakeholder identification. You move on to requirements discovery. You envision requirements as you collaborate with your stakeholders, and create a ranked work item list of functional requirements, nonfunctional requirements, defects, and so on. You begin writing your user stories and elaborate on the requirements as you progress in your construction iterations.

Unfortunately, most IT projects aren't green-field. Most time and effort is spent maintaining existing systems that are operating in an always-on enterprise. Those maintenance projects can be quite complex, because you must honor existing system interdependencies and avoid system downtime. Moreover, IT shops are now frequently asked to substantially modify existing assets to deliver new capabilities that address an array of requirements, such as those driven by competition, government regulations, customer needs, or internal pressures. And "these requirements need to be addressed today, please!"

Success in this higher-pressure world requires, among other things, more seamless collaboration between IT Development and Operations departments. Here's where agile practices can help. The focus of this article is on agile-based process improvements necessitated by this requirement for higher-bandwidth collaboration between IT Development and Operations. My definition of "Operations" includes infrastructure, ongoing support, maintenance, and improvement of a product. I discuss agile requirements practices that can help improve IT's ability to excel in this current, more challenging and changing world. I also explain how to leverage the role of the business analyst (BA) to bridge the gap between Dev and Ops.
Sources of change
Maintaining an existing product or system is the process of refining and improving to make sure it continues to meet business needs. Potential changes are submitted to the product manager in the form of change requests. Changes can be small or big. Changes that require significant effort are typically handled as a new feature and follow the new feature process. Changes can come from multiple sources, and they can be treated differently, depending on the source of the request.

• The most common source of problem reports is the Operations group that identifies defects and errors in the product that must be fixed. These change requests are usually given immediate attention because they can cause significant problems.
• The second most common source of change requests is enhancements requested by users. Because satisfying the users is important, these requests are often given priority, but typically second priority after major errors or bug fixes. Enhancement requests can include these examples:
  • New features to be added to an existing product or component
  • Modifying existing functionality, that is, modifying user stories to comply with updated business rules, legislation, marketplace change or new constraints
  • Building extensions onto a commercial off-the-shelf (COTS) system
• Third, changes can result when the underlying software, network, or operating system changes (for example, Android vs. Microsoft Windows). These adaptations often require changing the way that an application interacts with the new operating system or new software. Although these changes might be invisible to users because they do not affect functionality, they often affect the business or technical needs of the enterprise. These changes can be among the most challenging to implement and can impact the original set of nonfunctional requirements that have been defined and implemented already.
• A fourth source of change requests is executive and senior management. These change requests are often driven by major changes in the organization's strategy or operations. These changes are often treated as separate projects, but usually, the product manager who is responsible for the initial product is put in charge of the new project and becomes the product champion (as discussed in Discover to Deliver, cited in Resources). In DAD, this role can be filled by two people: an architect owner with the technical expertise and a product owner who has business analysis skills and the knowledge of the business domain.

However, we cannot ignore that there are challenges when it comes to making changes and in maintaining or supporting a product when the development team used an agile approach. Regardless of the organizational structure and whether you have Development separate from Operations (Dev and Ops) or a joint group (DevOps) all stakeholders need to be accountable for a successful usage and operation of a product.

The challenges of collaboration
At the start of any new agile development project, the team tends to choose the appropriate process and practices for their situation. They typically avoid a significant amount of bureaucracy, develop a "just about enough" set of artifacts, and legitimately do not think of what will happen after the product is handed to Operations. Yet the team that is operating a system needs to understand how it was built, what were the initial requirements (functional and non functional)...
and the criteria for prioritization. This will make their work easy if they need to agree on any new enhanced features made to them later. Unfortunately, Development and Operations teams are usually two separate entities, and they seldom collaborate and share knowledge. The following categories cover the key challenges.

Documentation
Maintaining an existing product without adequate documentation from the Development team is challenging. An agile team might have held all of the technical requirements in their heads. Maybe all you have is the user stories from the initial backlog and the code that you have implemented. The main critique against writing user stories simply and succinctly is that they lack sufficient details to be fully understood by the Operations team who will have to agree on further changes to make to the existing system.

Stakeholders
In an agile project managed using agile planning techniques, the backlog is the main artifact for scope and change management. When changes happen, they are added to the stack of items and prioritized in the same way as the rest of the items, by adopting a risk-value lifecycle practice (see Disciplined Agile Delivery, cited in Resources). When the prioritization occurs, the product owner must represent all stakeholders. If you do not have Operations and support staff represented, it becomes impossible to agree on the scope and goals of the effort, including those for Operations. In addition, the user stories and your backlog items do not give the designers a context to work from.

Persistence of requirements
We usually omit making our requirements persist in the acceptance test suite. This is related to the acceptance test-driven development practices usually performed to drive the code from the acceptance tests and not very well-understood by the agile community. In addition to lack of persistence, when requirements models are produced, they are often thrown away after the code is developed, so there is no record of how the requirements were designed and implemented. This is what DAD advocates doing.

Traceability of requirements
DAD and other agile approaches do not provide guidance on traceability of requirements. If a product is maintained, and if you do not have traceability data available, it becomes challenging to make changes during maintenance and trace back to the original requirements of various types or to tests that have been used during development.

Automation
Some organizations are comfortable with storing their backlogs in word-processing documents, and they do not have a way to manage versions and changes in their stack of work items.

Business analysts
In IT, we tend to recognize that developers also work on maintenance projects. However, there is little in the agile literature about the role of a business analyst (BA) or product owner
who faces substantial changes to a product following its initial release. The agile community has not invested much time and effort exploring how to handle requirements and changes after the product is released, for an overall DevOps solution.

To summarize this section, we know how hard it is to maintain a system without adequate requirements and architecture or design documentation. Using the option of exploring a black box has already been proven to be time-consuming. In addition, the Operations team does not work long enough with the development team to understand the requirements for the product being built, and collaboration is often kept to a minimum. Further, activities within the Operations department tend to fall somewhere outside of the BA’s role.

Next, let's go through a change scenario to better understand what can fall under the responsibility of a BA when a product is enhanced or new features kick in.

**A sample scenario for change**

The diagram in Figure 1 helps illustrate the various change scenarios. The opaque box in the diagram represents the system under development. If you are lucky, you have requirements documentation that can provide visibility into the system.

**Figure 1. A black box representing a current product**

As the BA or product owner for the project, you have been asked to add new functionality for the E region. How do you go about dealing with this additional functionality? When you add new features, you might be required to change some screens, add new UIs, or add new data fields. This proposed change is represented by the grey space in the diagram. How do you get to know what needs to be changed in the grey space? How do you integrate E into the existing product?

What about the "connectors" that will help tie the new features to the existing system? In an agile environment, BAs should become involved in requirements management for continuous maintenance upgrades of already-deployed software. Over time, the systems that we develop using an agile approach become larger and larger, and the role of a BA becomes prominent. Projects using scrum methods are usually so well-contained that communication among all stakeholders is not a big issue. The agile team members usually have access to each other, so it is easy to clarify ambiguities. However, as projects grow in scope and size with distributed teams, the communication pathway becomes more complex. To succeed in a DevOps environment, we
need to adopt a DAD approach and bridge the gap between the Development and Operations disciplines. What is the minimum a BA can do in agile development and that will help improve the ability to perform future maintenance to the software?

**The key element for DevOps: A business analyst who understands the big picture**

As Scott Ambler pointed out in a presentation at the Rational software Innovate conference in 2012: "An important consideration for an effective DevOps strategy is to ensure that your development teams produce solutions that work well with your Operational environment."

Today, the scale of IT is massive, business applications evolve at a ludicrous speed (Facebook, Google), and development teams push new changes every day. Discouraging change might not be the right way to do business, but you have to be smart about how you encourage change.

BAs become crucial in DevOps. They are in the primary role for dealing with new changes or requests for enhancements (RFEs) as they arise. As a BA, it is important to remain informed of all changes to the initial product and if needs be wear an Operation duty hat to keep the house in order.

Development and Operations stakeholders have different priorities, and the BA is the bridge. Development teams want to fix the problem themselves rather than wait for Operations to make a decision to handle the problem. Operations is perceived by Development as the "change blockers," and Development is perceived by Operations as "a bunch of hackers." Generally, this leads to conflicting goals. Involving a business analyst helps mediate this potential conflict in several ways:

- BAs can help align Operations and Development by unifying and extending the agile requirements practices through the full lifecycle, including Operations.
- When BAs are part of the team, they can help overcome the challenges in reaching agreement on requirements among stakeholders, including Operations staff.
- Good collaboration between BAs and Operations prevents developers from making ad hoc changes or adding new features ("gold plating") without customer approval.
- Operations wants to keep the site stable to enable the business in the best way. Stability, reliability and other non functional requirements are common requirements that a BA should be aware of. Therefore, when the business or the architecture team requires changes, BAs and Operations should be included in the discussion about those requirements, along with Development.

Having a BA on the team to coordinate and cultivate a good working culture of gaining consensus among the stakeholders and managing changes adds to the success of DevOps. However, this alone does not ensure success. If you need to produce solutions that work well with your operational environment, you must also consider scaling the agile requirements practices to satisfy the needs of DevOps.
A DevOps view of agile requirements practices: It’s all about scalability

Let’s enter the world of the agile requirements practices and see how they can be scaled and adapted to the needs of the DevOps approach.

**Adopt a robust shared vision**

Identify all key stakeholders from both sides (Development and Operations). Use a Vision artifact to express who the key stakeholders are and to convey their needs in business and technical terms. For DevOps to be successful, development teams must accept that operations and support staff are key stakeholders with whom they should work throughout the project.

**Adopt scaled requirements envisioning**

Use these additional techniques and tools:

- Keep track of some of the key requirements artifacts for posterity. For most cases, posterity is not further away than your release 2 or release n that your team or another team will have to work on. As you consider enhancement in the next releases, one key artifact that is easy to develop is a use case diagram. For DevOps solutions, creating a use case diagram from a list of user stories adds context and provides a good way to verify that all of the functional requirements are covered. DAD is a usage-driven approach, and having an artifact such as a use case diagram that adds value is not anti-agile.
- Explore diagramming for viewing the use of the solution. Models such as a use case diagram, a sequence diagram, and others that have been used to supplement the user stories, and were valuable to the team at the time when the details of the user stories were fleshed out, are worth keeping and should not be discarded. I'm talking about persistence again here.
- Adopt a "requirements map" for traceability between artifacts throughout the full lifecycle. This helps in reasoning about the world of requirements and its link with the rest of the testing and implementation world. Even if some valuable documents have been produced (for example, a user manual), without requirements traceability, the value of those artifacts will not be realized. Building a pictorial requirements representation helps you see the business needs and requirements not in isolation but in the context of the importance of the full lifecycle of the application, including for DevOps solutions. For this reason, the linkages between development, testing, and requirements are the most critical and most useful when it comes to understanding the product being developed.
- Automate requirements analysis and management — this is key! When possible, use a tool such as IBM® Rational® Requirements Composer to keep track of your visual requirements models and linkages between artifacts. Speaking of linkage, one key capability that is useful for future maintenance is the "link explorer" map. It's the one artifact to have, when you can only have one. Figure 2 is a sample requirement map that shows the linkage between a feature and the rest of the artifacts developed through the lifecycle.
Figure 2. Visual traceability in the Rational Requirements Composer link explorer

- The diagram includes all of the ramifications and connections to existing rules, tests, storyboards, other constraints, and implemented stories. Building a requirements map provides visibility and puts an agile team in a position to perform future enhancements more efficiently and increase the chance of delivering a maintainable product. As Karl Wieger stated in More about Software Requirements: Thorny Issues and Practical Advice, although it is difficult to quantify, this investment will pay dividends anytime that you have to modify, extend, or replace the product. Indeed, such a representation facilitates reasoning and keeping momentum between testing, development, and requirements for future maintenance.

Adopt a scaled iteration modeling practice

Do this during the construction phase. Models and diagrams incorporate knowledge of the product under development. Doing just enough modeling and keeping a comprehensive set of diagrams and sketches increases the chances of delivering a maintainable product. Later, Operations can reuse them if any reverse-engineering analysis of the part of the product that you are modifying is required. Use requirements modeling tools to store your requirements information that is captured in multiple models: Making those important models persistent is key.

Adopt team change management practice

A disciplined, agile, lightweight change management process that helps you track all changes to your original release backlog and your requirements maps can reduce disruption caused by changes. Simple changes often have far-reaching impact, requiring that many parts of the products be modified. As a result, adopting this practice and using a tool to track potential impact provides transparency, and everyone can see what is going on. In addition, this gives you an effective, agile process to use on your maintenance activities.

Adopt acceptance test-driven development

It is worth taking time to document your acceptance criteria and to adopt an acceptance testing story approach. Even though the agile community understands that requirements and acceptance tests are not separate and they should be coupled, this practice is not yet mature. DAD recognizes that you need a requirements specification but also recognizes that your requirements specifications are not separate from the acceptance tests, unit tests, and others. Acceptance tests record the decisions made during the course of the requirements discovery and when elaborating on the stories. Further, acceptance tests are the ones that persist within the team’s automated or manual regression test environment. Writing acceptance tests is lean thinking, because there
Adapting agile requirements practices to ongoing enterprise improvement efforts: DevOps to the rescue

is a more systemic view of the requirements across the team in lean development methods, and testing is not separate from requirements. This is an opportunity for DevOps solutions to further explore the benefits of applying this practice in an agile environment.

**Adopt an overall agile governance and a scaled full risk-value lifecycle practice**

These approaches support the agile requirements practices for DevOps in several ways:

- There is a degree of governance necessary to guide the DevOps solutions. Agile-appropriate governance, supported by the right requirements metrics and a goal-based approach, is required to bring the work of Development and Operations teams closer. In this goal-based approach to commitments, both teams are accountable for collaborating closely to aim for common deliverables and joint process improvement. Building this trusted partnership between the two teams minimizes the time and cost spent on knowledge transfer.
- Further, to enable explicit governance, DAD promotes enterprise awareness, which is about leveraging existing assets, following and adopting common policies and guidelines, and following existing enterprise conventions. Aiming for common artifacts is necessary for a successful DevOps solutions.
- To reflect the reality, scalability is another thing that you get with DevOps. The DAD full lifecycle has three phases (Inception, Construction, and Transition), and it is evolutionary. To cater for DevOps solutions, it can be extended to include post-delivery activities to assist with support and Operations. However, it is not enough to have the feedback loop in the DAD full lifecycle, from the end of a release to the Development stack of work items at end of the inception (see *Disciplined Agile Delivery*, cited in Resources). Instead, you need to account for the Operations staff roles during the lifecycle of the product and the Operations activities and production teams' own goals to reach their own milestones. When you transition your product to be delivered to your end users, at what point do you commit to Operations, and support? To address the needs of DevOps solutions, it is worth exploring the addition of a separate phase to extend the existing DAD lifecycle and add other explicit milestones or light reviews related to Operations. Would that be different if we have a single organization, DevOps? I'm not sure.

**Adopt requirements refactoring**

Refactoring is the practice of continuously making incremental improvements in your system without changing the overall external behavior. Within the agile community, we tend to believe that refactoring applies only to architecture and design. Unfortunately, even for these two disciplines, some organizations tend to believe that it is costly to refactor, so they do not encourage their teams to apply this practice. DevOps definitely increases the importance of these refactoring practices at the design level but also at the requirements and testing levels. Some new features might contain constraints when requirements are specifying enhancements to be made to an existing system. Before adding new features built on older ones, cleaning and improvement on already existing models, diagrams, and test cases might be required for better readability and maintainability. Changes must integrate smoothly with the existing reality, and refactoring requirements to take into account such additional constraints is natural in these situations. For
example, you do not have to write stories from scratch, but you can reuse and improve the existing
requirements sets in any of these ways:

- Rename user stories. Let's suppose that you added acceptance criteria to an existing story.
  Rather than adding a new user story, you can rename it to allow for the newly identified
criterion.
- Split some of the stories.
- INVEST more in improving the existing requirements to avoid dependencies with the
  additional ones, in the ways that Bill Wake (see Resources) defines INVEST: Independent,
  Negotiable, Valuable, Estimable, Small, and Testable.
- Rename storyboards, UI sketches, and so forth. Even if those improvements are commonly
  made during the agile development lifecycle, people tend not to spend much time on
  improving names.

Summary

At the low end of the formality scale, scrum project management methods and Extreme
Programming (XP) have no explicit documentation practice. These approaches are useful when
software projects remain simple and are executed by a co-located team. User stories are the
primary vehicle for carrying the customer's requirements through the value stream, from discovery
to just-in-time (JIT) analysis, through code, testing, and deployment. In such smaller, well-
contained projects, the lack of requirements documentation is seldom an issue. In fact, agilists
argue that stacking all requirements in the backlog as user stories avoids additional documentation
and other administrative overhead and facilitates frequent interactions among the whole team and
stakeholders. It is also claimed that this is a better way to ensure a complete understanding and
translation of requirements.

However, as project complexity grows in size and distribution and as compliance needs and
the importance of nonfunctional requirements increases, a Disciplined Agile Delivery approach
becomes crucial to project success. DAD bridges the gap between requirements and development
better by leveraging a full risk-value cycle, adding agile governance, and specifying the adoption
of several complexity-driven practices in the architecture and requirements. In such environments,
DAD is a better fit as a foundation for DevOps. DAD and tools alone won't save you, though.
People must be convinced to buy into the cultural part of DevOps. But that's a topic for another
article.

© Copyright IBM Corporation 2012
Trademark
(www.ibm.com/developerworks/ibm/trademarks/)