Scaling Software Agility

Best Practices for Large Enterprises

Dean Leffingwell

Foreword by Philippe Kruchten

The Agile Software Development Series

Alistair Cockburn and Jim Highsmith,
Series Editors

Scaling Software Agility

Small Chunks Uncover Key Risks Early It is not unusual for teams new to agile to achieve only 25 to 35 percent of the scope of work they set out for themselves in the first iteration. The reasons can be varied, as we'll see in later chapters. But whether the goal is missed because, for example, the teams are unable to estimate very well or perhaps because a small story uncovered an unanticipated technical obstacle, the results of the first iteration will immediately expose the risks, wherever they may lay. In addition, *management has its first tangible view of progress at the end of only the first week or two of the project.*

SUMMARY

We've seen in this chapter that, conceptually, agile is indeed very different from our plan-driven methods. However, given this data, we can now also start to imagine how these different paradigms can potentially deliver dramatically different results. That is why we are here, and that is the power of agile.

This page intentionally left blank

Chapter 8

The Challenge of Scaling Agile

Barriers to agile adoption at the enterprise arise from two sources: the apparent limitations of the methods themselves and the impediments that exist within the enterprise. Both must be addressed to achieve enterprise agility.

Applying what we have learned so far about agile, in the next few chapters we look at a number of best practices of agile that natively scale to the enterprise level. It is of great convenience to the enterprise that even deploying a small number of heads-down agile teams applying these best practices can create substantial improvements in productivity and customer satisfaction at the local team level. At enterprise scale, however, the challenge of achieving the full benefits of agile is significant, and CIOs, VPs, and other sponsors should recognize the likelihood that serious challenges in the existing organization must be addressed.

The fact is that many enterprises that depend on software are no longer very good at developing it. As the enterprise grows, organizational patterns, policies, and procedures grow with it, and some, perhaps many, may run directly counter to the philosophies that characterize agile. These patterns resist change, and yet change is necessary to unleash the creative power and productivity of the software practitioners they employ.

Moreover, when it comes to scaling agile to the enterprise, there are two classes of challenge that must be addressed. The first—the challenges inherent in agile itself—presents the apparent limits of the technology because of the fixed rule bases and assumptions built into the methods. The second—those imposed by the enterprise—are impediments that likely exist within the enterprise that will otherwise prevent the successful application of the new methods. *Both* types of challenges must be successfully addressed in order for the enterprise to achieve the full benefits of agile.

APPARENT IMPEDIMENTS OF THE METHODS

While the thesis of this book is that, with modification and adaptation, agile *does scale to the enterprise level*, the fact remains that the methods were developed and codified in small team environments where there was freedom to explore and innovate, and where most resources the teams needed and most problems that arose could be managed at the level of a single team.

In addition, we must recognize that many successful applications of these methods at somewhat larger scale (dozens of developers using XP, Scrum, etc.) occurred in a context where there were many small and somewhat autonomous projects inside the enterprise wherein specific projects aligned well with the principles of agility and most of the rest of the enterprise was not dependant on components, subsystems, features, or whatever else these teams were delivering. These applications could have been smaller, standalone products, or internal applications, or perhaps new Web front ends for legacy applications; but in any case, they were relatively isolated endeavors that did not require coordination of large numbers people or other teams and other departments. The basic small-team constructs of agile were inherent attributes of the project, and the organization did not interfere with success. However, when building enterprise class systems, systems of systems, and applications including components and enterprise systems provided by others, the apparent limits of the methods themselves must be addressed. These limits include small team size, close customer involvement, collocation, emerging architecture, lack of requirements analysis and documented specifications, and culture and physical environment.

Small Team Size

XP and Scrum recommend teams of eight or fewer people, including product owner/managers or other customer proxy, developers, and testers. In many cases, teams decide that even that number is too large, and they may subdivide into component or feature teams of three to five people. To an enterprise with 1,000 practitioners to deploy, thinking in terms of managing literally hundreds of teams boggles the mind, and there arises the need to understand how to fit this new model into the existing organizational hierarchy.

Customer Is Integral to the Team

We've witnessed XP success (in particular) in a number of IT-like environments where the customer was literally down the hall and the business analyst who worked with that department was willing and able to participate in

the fine-grained, detailed review that stories and customer-written acceptance tests require. For many enterprises, however, this is not the case: the customer may be remote or may not have the skills or time available to participate in such a manner. Or perhaps the enterprise is a large ISV (independent software vendor) where the application has tens of thousands of users and where there is no single customer to satisfy. Every customer is different, and the larger ones speak loudly indeed, and the information flow is attenuated through many groups before it reaches the team. And while product managers are likely playing the proxy role, it is very unusual to find sufficient, tactical product management resources already present that can drive the tactical detail required by fast iterations and rapid response to change.

Collocation

Much of the productivity of agile comes from the pairing contexts, daily stand-ups, visual signaling of stories and status, and constant informal communication that characterizes these methods. Developers, product owners, and testers are together, not separated by time zones and language barriers. At scale, collocation is impractical even for large teams in the same building, and other mechanisms must be devised. Moreover, it is likely that many team members are in different countries and different time zones and perhaps even speak different languages. *At scale, all development is distributed development*, and the methods and organization must evolve to address this challenge.

Architecture Emerges

Agile, and more specifically, XP, trades off the ability to refactor code quickly with the cost and investment necessary to create a more forward-looking architecture, some *architectural runway*, that coordinates the efforts of distributed teams. With larger scale systems, however, the refactor cost curve discussed in Chapter 3 may not be the case, because hundreds of person-years may be necessary to get even a first release deployed. Since agile generally does not coach how to approach architecting larger scale systems, that apparent limitation is often a real impediment for agile adoption. In addition, the role of system-level architects is not prescribed by agile, and that is a cause for concern for those (including real system-level architects) who understand that systems of scale cannot be successfully built without solid architectural underpinnings. Many agile initiatives have been stopped at the door with the simple response, "For systems of our complexity and scale, we do not believe that architecture emerges. So take your lightweight methods with you as you go."

Lack of Requirements Analysis and Documented Specifications

Agile's practice of working on a few stories at a time is a wonderfully focusing mechanism for the team. But in larger systems, what drives these stories into existence? Who says these stories are the right stories? Will the summation of all these stories (now in the thousands) actually meet our customer's end-to-end use-case needs? Does our team's product owner have clear visibility into stories others are building? Are they likely to affect us? If so, when? And when developing solution sets (large sets of products that must be deployed together and support end-to-end use cases for the user or customers), how do we know that the stories on the table actually work together to achieve the final objective? Can building an enterprise application, one story at a time, possibly work? Well, perhaps not exactly that way.

Culture and Physical Environment

Effective agile environments look and sound different. Most team members work in an open, or nearly open, environment, and often their managers and supervisors have left their offices or cubbies and joined the team at those tables. Also, the process doesn't look very efficient because there is a certain informality to stories posted on walls, whiteboards shoved hither and yon, and teams of people seeming to do far more talking than coding.

To some, this scenario may appear to be unsupervised, chaotic, and even unprofessional because the teams do not seem to bear the trappings we come to expect in an efficient, well-organized enterprise. We've also observed that, for whatever reason, the dress code on agile teams tends to be somewhat more relaxed than the standard business-casual fare, and the combination of casual dress, pairing, constant communication, and stuff stuck all over the walls does not fly well in some corporate circles.

IMPEDIMENTS OF THE ENTERPRISE

Compounding the limitations just discussed, the enterprise itself often brings some baggage of its own to the "table of agile progress." As we well know, the enterprise is a living organism, and as such, it long ago learned how to defend itself—and its defenses for the existing status quo are likely to be formidable. Typical impediments to change include the following.