Beyond Scrum + XP: Agile Architecture Practice

by Ipek Ozkaya, Robert L. Nord, Stephany Bellomo, and Heidi Brayer

The belief that Agile requires small colocated teams, downplays architectures, and delivers no documentation still prevails among many software practitioners. The reality is that organizations, in their quest to rapidly field projects, are building on the strengths of Scrum and XP. They are doing so to creatively combine Agile architecture practices to achieve the benefits of Agile across the lifecycle.

We found further evidence of this more expansive approach when we interviewed representatives of five government and commercial organizations that operate in highly regulated settings.[1] Our goal was to gain a better understanding of success and failure factors in rapidly fielded projects using Agile software development practices. The reality of a highly regulated environment is that practitioners must often balance the demand to quickly deliver functionality with a desire for a stable, reliable system, especially when faced with the challenge of sustaining such systems for several decades. In addition, in highly regulated environments such as avionics, financial services, and healthcare, software development teams need to interface with system engineering, deployment, and quality assurance teams that may be operating under different development and delivery tempos. These competing pressures often result in projects marked by high initial velocity followed by a slowdown that results from stability issues (see Figure 1).

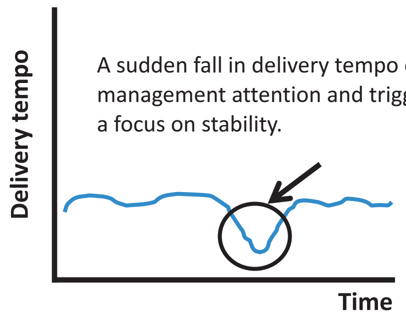

Figure 1 — A common pattern of delivery tempo in support of rapid fielding.

The interviews showed that most experienced practitioners, when faced with challenges, did not apply Agile practices in a silo. Rather, they used their expertise to creatively combine them with other practices, especially architecture, to respond effectively to stability issues while rapidly fielding projects. Doing so helped them avoid significant disruptions in velocity. The experiences of these organizations support the stance that a more expansive application of hybrid practices is not only necessary but essential in balancing the opposing objectives of speed and stability.

In this article, we highlight several approaches applied by these organizations and provide a more in-depth look at two of the practices: release planning with architecture considerations and roadmap planning with external dependency management.

SPEED VS. STABILITY

The essence of balancing speed and stability involves achieving and preserving a software development state that enables teams to deliver releases that stakeholders value at a tempo that makes sense for their business.

The desired software development state is different for each organization and needs to be understood clearly. This is a state in which architecture (often in the form of platforms and application frameworks), supporting tool environments, practices, processes, and team structures exist to support efficient and sustainable development of features. The entire organization, including development teams, management, and stakeholders, must have visibility into the desired state, so that they neither overoptimize the supporting development infrastructure nor quit working on it.

We asked senior software developers and managers to describe factors that either enabled or inhibited the speed of delivery and the degree of stability in the software product. The factors fall into one of three common situations:

1. When the project was going well, teams applied foundational Agile practices commonly touted as "enablers of success," such as daily Scrum meetings,

a Scrum collaborative management style, continuous integration, test-driven development, and so on. Small dedicated teams are able to stay within bounds due to the well-known nature of the infrastructure and the limited scope of the project. Teams facing issues of scale and complexity are able to stay within bounds because of awareness across the organization of the need to maintain the infrastructure.

2. When teams encountered a problem that was taking them away from their desired state, they would often combine Agile practices with architecture and other disciplines, such as management and engineering, to make incremental adjustments to ensure they had sufficient technological infrastructure to support development.
3. When teams encountered problems that were not visible to management and stakeholders, the adjustments were disruptive. Solutions were delayed until the chronic problem became visible. In certain cases, it was not possible to adjust course because the problem became visible too late; the project was not able to deliver, and the team failed.

HYBRID PRACTICES THAT ENABLE FAST, STABLE DEVELOPMENT

Our interviews revealed the following examples of Agile architecture practices that enable speed and stability:

- Release planning with architecture considerations
- Prototyping with a quality attribute focus
- Roadmap planning with external dependency analysis
- Merging of test-driven practices (e.g., automated test-driven development and continuous integration) with a focus on runtime qualities (e.g., performance, scalability, and security)
- Technical debt monitoring with a quality attribute focus

These Agile architecture practices allow more experienced practitioners to avoid project slowdowns related to stability issues with minimal disruption to capability delivery. While these practices have been advocated for a while, using them within the confines of a well-defined process, such as Scrum, becomes challenging.

We will now describe in more depth two of these practices that all five organizations used: release planning with architecture considerations and roadmap planning with external dependency analysis.

Release Planning with Architecture Considerations

Development teams often incorporated architecture considerations into release planning in response to problems associated with prioritizing features visible to the stakeholder. One organization adopted the Scrum release planning management process without much visibility into the infrastructure needs. The increasing focus on rapid delivery inevitably made the organization realize that its teams needed to work in parallel to meet schedule demands. So the business moved from a centralized development model to a geographically distributed work model. Delivery slowed, however, because there was not enough architectural definition in the feature documentation to allow the teams to work independently. This triggered a closer look into the infrastructure and a more stable architecture. The outcome was to incorporate more explicit infrastructure and architecture planning into release planning and not simply focus on the high-priority features.

The important and unexpected observation revealed by our interviews is that all the organizations recognized that without incorporating architecture into release planning, it is not possible to achieve the expected delivery tempo after significant and unexpected change. In all cases, the projects initially appeared within bounds of their desired state, and issues were visible. It was only after the disruption that the teams sought deeper visibility into the project.

Roadmap Planning with External Dependency Analysis

One organization incorporated external dependency analysis into its roadmap planning process. This approach reduced the risk of being blindsided by unanticipated conditions due to dependencies on expertise external to the team, infrastructure components governed by other parties, or difficult-to-reach users. For instance, during the development of an operational sprint, several firewall ports governed by an external party were closed without notice, causing sporadic stability issues that were difficult to troubleshoot. Until this event, the dependency on security decisions governed by another team had not been realized as a critical dependency that could impact the development effort.

Since this problem was holding up development, the team took immediate action to analyze and reassess external dependency risks that could affect their design decisions and development. Team members then devised a mitigation strategy for each risk. Some strategies required modifications to the change management notification process, and others required a deeper understanding of dependencies on components being developed by other teams. The roadmap document, which contained a description of development by phases, was used to capture external dependency risks and mitigation strategies at the portfolio level. The team incorporated the change into the ongoing practices with limited disruption to workflow; therefore, it followed the incremental response cycle. The team also adopted the practice of continuing to conduct external dependency analysis regularly to identify external dependency risks at the roadmap level.

The criticality and impact of technical dependencies intensify at scale. Such dependencies are also easy to overlook because they may not be exercised daily or even at each sprint. Therefore, the roadmapping level is the right place to surface and track these dependencies. It allows the architectural decisions to propagate correctly among software elements as a system is developed in breadth and depth and among the multiple development teams across an organization.

INHIBITING FACTORS

The factors that prevented development teams from rapidly delivering the software product, or mired them in a state outside the bounds of acceptable software development, included often-observed inhibitors such as slow business decision-making processes, limitations in measuring architectural technical debt, overdependency on the architect for architecture knowledge, and stability-related efforts not entirely visible to the business.

The inability to deal with scale and complexity also emerged as a factor. Development teams from four of the five organizations described situations in which they were not able to complete test cases within the targeted iteration due to increasing software complexity and limitations in expertise and/or tools. They also reported that excessive focus on speed, and difficulty in making architectural problems visible to the business side of an operation, often led to major redesigns or bug-fixing sprints.

Many of these negative outcomes are the result of inconsistent and incorrect applications of Agile and/or architecture practices. Several factors can be traced back to their enabling counterparts. For example, a desire to quickly deliver features caused stakeholders to overlook the importance of stability and limited requirements analysis and stability-related work. Demonstrating the criticality of stability required improved measurement of technical debt. Two of the organizations took actions to improve their visibility into technical debt by tracking it explicitly in their backlogs.

Organizations acknowledged the tradeoffs they experienced when taking shortcuts in software development to accelerate delivery. They are now more aware of how degraded quality leads to technical debt and are taking steps to address it. In response to business pressure, organizations sometimes embedded architectural changes within unrelated features during development. This lack of transparency can result in incorrect productivity measures as well as unanticipated schedule impacts. They expressed the belief that if they were able to make technical debt more visible to stakeholders, they could avoid potentially costly and disruptive changes in favor of more incremental changes supporting a sustainable delivery tempo. We call this "technical debt monitoring with a quality attribute focus."

TRANSITIONING ENABLERS OF CHANGE

In a recent blog post, Ken Schwaber said he would like to change the mindset of "Scrum But" to "Scrum And."[2] He explained that the use of "Scrum And" characterizes an organization that is on a path of continuous improvement in software development beyond the basic use of Scrum. He gave this example to illustrate the concept of extending Scrum: "We use Scrum, and we are continuously building, testing, and deploying our increments every Sprint." The experiences described by organizations we interviewed support the stance that practice extensions are needed and anticipated in iterative and incremental development. Evidence from Guest Editor and Cutter Senior Consultant Scott Ambler's Agile project initiation survey has also shown consistent results with the experience of the teams we interviewed.[3]

In this new "Agile And" world, some organizations are moving toward Disciplined Agile Delivery (DAD), a hybrid, people-focused IT approach developed by Ambler. In line with our observations from practice, such new-generation software development practices place the emphasis on a full delivery lifecycle that is process goal–driven and incorporates architecture and enterprise awareness.[4] Others are embracing the Scaled Agile Framework (SAFe) developed by Dean Leffingwell and colleagues, not only incorporating architecture explicitly into the development lifecycle, but also Lean methods and portfolio management.[5]

At the SEI, we work with organizations that must operate in a climate of ever-shrinking budgets coupled with near-constant demands for new capabilities. We advise organizations looking to adopt or enhance Agile practices in their pursuit of rapid delivery to begin by conducting a review of architecture-centric risk factors for adoption of large-scale Agile software development. While Agile architecture practices can help these organizations ensure the stability of the systems they are fielding, it is important to understand the root causes of the inability to deliver at the expected pace and how the tension between speed and stability is managed. Organizations must also make the problems more visible to developers, management, and stakeholders.[6] When considering how to combine Agile and architecture practices, organizations must first ask the following questions:

- Are we delivering software to our customer at an expected pace?
- Are we aware of problems that are cropping up as a result of losing focus on architecting when Agile adoption activities become the primary focus?
- Does our technical roadmap address short-term and long-term issues?
- Does the team of software developers have skills that would enable them to successfully implement Agile and architecture?
- Do we have visibility into not only the project management of the system, but also the quality expected from the system?

We hope that by codifying and sharing the practices described above, other organizations can learn to apply these approaches to contend with the demands of rapidly delivering software that is reliable, stable, and flexible in a fast-changing environment.

ACKNOWLEDGMENT

This material is based upon work funded and supported by the US Department of Defense under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

ENDNOTES

[1]Bellomo, Stephany, Robert L. Nord, and Ipek Ozkaya. "A Study of Enabling Factors for Rapid Fielding: Combined Practices to Balance Speed and Stability." Paper presented at the *International Conference on Software Engineering (ICSE) 2013*, San Francisco, California, USA, 18-26 May 2013.

[2]Schwaber, Ken. "Scrum But Replaced by Scrum And." *Telling It Like It Is*, 5 April 2012 (http://kenschwaber.wordpress.com/2012/04/05/scrum-but-replaced-by-scrum-and).

[3]Ambler, Scott W. "Agile Project Initiation Survey Results: July/August 2009." Ambysoft, 2009 (www.ambysoft.com/surveys/projectInitiation2009.html).

[4]Ambler, Scott W., and Mark Lines. *Disciplined Agile Delivery: A Practitioner's Guide to Agile Software Delivery in the Enterprise*. IBM Press, 2012.

[5]Leffingwell, Dean. "Scaled Agile Framework" (http://scaledAgileframework.com).

[6]Gagliardi, Michael, Robert L. Nord, and Ipek Ozkaya. "Architecting for Large Scale Agile Software Development: A Risk-Driven Approach." *CrossTalk*, May/June 2013.

Ipek Ozkaya is a senior member of the technical staff at the Carnegie Mellon Software Engineering Institute (SEI). With her team at the SEI, she works to help organizations improve their software development efficiency and system evolution. Dr. Ozkaya's work focuses on software architecture practices, software economics, and requirements management. She serves as chair of the advisory board of IEEE Software *magazine and as an adjunct faculty member for the Master of Software Engineering Program at Carnegie Mellon University (CMU). Dr. Ozkaya holds a PhD in computational design from CMU. She can be reached at ozkaya@sei.cmu.edu.*

Robert L. Nord is a senior member of the technical staff at the SEI, where he focuses on Agile and architecting at scale and works to develop and communicate effective methods and practices for software architecture. He is coauthor of the practitioner-oriented books Applied Software Architecture *and* Documenting Software Architectures: Views and Beyond*, and he lectures on architecture-centric approaches. Dr. Nord is a member of the steering committee of the* WICSA Conference Series*, in addition to organizing events at software engineering, Agile, and architecture venues. He earned a PhD from CMU and is a distinguished member of the ACM. He can be reached at rn@sei.cmu.edu.*

Stephany Bellomo is a senior member of the technical staff at the SEI, serving in the Architecture Practices group and the Value-Driven Incremental Development research team. Ms. Bellomo is currently focused on work in the area of Agile and architecture practices. Prior to that, she focused on other areas such as service-oriented systems and Agile and information assurance. In support of this work, she has authored papers/reports, delivered presentations, served on panels, and provided onsite technical support. Ms. Bellomo was Tutorial Chair for the 2013 SEI Architecture Technology User Network (SATURN) Conference *and a technical reviewer for submissions to the* SEI Software Product Lines *conference. She teaches the SEI courses "Service-Oriented Architecture Migration of Legacy Components" and "Software Architecture Principles and Practice." Ms. Bellomo received a master's degree in software engineering from George Mason University. She can be reached at sbellomo@sei.cmu.edu.*

Heidi Brayer is a writer and editor at the SEI. Her work focuses on writing and editing a weekly blog to highlight SEI research initiatives. She also produces the SEI Podcast Series*, which is available on the SEI website and on CMU's iTunes U channel. She earned a master's degree from CMU's Master of Professional Writing program. She can be reached at hap@sei.cmu.edu.*